AFATL-TR-88-117, VOL I

Program EAGLE User's Manual

Vol I-Introduction and Grid Applications

Lawrence E Lijewski
John Cipolla, et al.

AERODYNAMICS BRANCH
AEROMECHANICS DIVISION

SEPTEMBER 1988

INTERIM REPORT FOR PERIOD OCTOBER 1986-SEPTEMBER 1988

AIR FORCE ARMAMENT LABORATORY
Air Force Systems Command ▮ United States Air Force ▮ Eglin Air Force Base, Florida

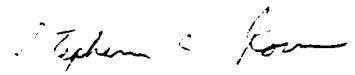
NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility nor any obligation whatsoever. The fact that the Government may have formulated or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise in any manner construed, as licensing the holder, or any other person or corporation; or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

The Public Affairs Office has reviewed this report, and it is releasable to the National Technical Information Service (NTIS), where it will be available to the general public, including foreign nationals.

This technical report has been reviewed and is approved for publication.

FOR THE COMMANDER



STEPHEN C. KORN
Technical Director, Aeromechanics Division

Please do not request copies of this report from the Air Force Armament Laboratory. Copies may be obtained from DTIC. Address your request for additional copies to:

Defense Technical Information Center
Cameron Station
Alexandria, VA 22304-6145

If your address has changed, if you wish to be removed from our mailing list, or if your organization no longer employs the addressee, please notify AFATL/FXA , Eglin AFB FL 32542-5434, to help us maintain a current mailing list.

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release,distribution is unlimited.
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S): AFATL-TR-88-117, Vol I
6a. NAME OF PERFORMING ORGANIZATION: Aerodynamics Branch, Aeromechanics Division	6b. OFFICE SYMBOL (If applicable): AFATL/FXA
7a. NAME OF MONITORING ORGANIZATION: Aerodynamics Branch, Aeromechanics Division	
6c. ADDRESS (City, State, and ZIP Code): Air Force Armament Laboratory, Eglin AFB FL 32542-5434	7b. ADDRESS (City, State, and ZIP Code): Air Force Armament Labortory, Eglin AFB FL 32542-5434
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: Aeromechanics Division	8b. OFFICE SYMBOL (If applicable): AFATL/FX
9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c. ADDRESS (City, State, and ZIP Code): Air Force Armament Laboratory, Eglin Air Force Base, Florida 32542-5434	10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO	WORK UNIT ACCESSION NO.
62602F	2567	03	08

11. TITLE (Include Security Classification)
Program EAGLE User's Manual
Volume I: Introduction and Grid Applications

12. PERSONAL AUTHOR(S)
LAWRENCE E. LIJEWSKI, JOHN CIPOLLA, JOE F. THOMPSON, BOYD GATLIN

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Interim	FROM Oct 86 TO Sep 88	September 1988	177

16. SUPPLEMENTARY NOTATION
Volumes II and III are provided by Mississippi State University
Availability of report on verso of front cover

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP
01	01	

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Numerical Grid Generation
Elliptic Grid Generation
Algebraic Grid Generation
Euler Solver
Transonic Flow
Implicit Algorithm

19. ABSTRACT (Continue on reverse if necessary and identify by block number)
This report documents the theory and usage of Program EAGLE; a three-dimensional, multiblock grid generation code and flow solver for arbitrarily shaped, advanced weapon airframe configurations.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED

22a NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL
JOHN CIPOLLA	(904)882-3124	AFATL/FXA

DD Form 1473, JUN 86 *Previous editions are obsolete.*

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

PREFACE

This report was prepared by Dr Lawrence E. Lijewski and John Cipolla of the Aerodynamics Branch (FXA), Aeromechanics Division, Air Force Armament Laboratory (AFATL), Eglin Air Force Base, Florida, and Drs Joe F. Thompson and Boyd Gatlin of Mississippi State University, Starkville, MS. The work was performed under Work Unit 25670308 from 1 October 1986 to 30 September 1988.

This report documents the usage of the Program EAGLE. The principal investigator of the surface and grid generation theory has been Dr Joe F. Thompson of MSU. The principal investigators of the flow code theory have been Dr David L. Whitfield of MSU, and Dr Dave M. Belk and Mr L. Bruce Simpson of the Computational Fluid Dynamics Section (AFATL/FXA). Capt Jon S. Mounts of the Computational Fluid Dynamics Section (AFATL/FXA) has increased the utility of the flow code through user-oriented inputs and outputs, extensive error checking, and calculation of component forces and moments. The program manager for the development of Program EAGLE has been Dr Lawrence E. Lijewski of the Computational Fluid Dynamics Section.

ACKNOWLEDGEMENTS

The authors would like to extend their appreciation to Lt Montgomery C. Hughson, 4FATL/FXA, and Mr William Riner of Sverdrup whose aerodynamic applications appear in this volume. Special thanks are also in order for our administrative assistant, Nancy Elliott, who diligently typed and proofed this report.

TABLE OF CONTENTS

Section	Title	Page
I	INTRODUCTION	1
II	JOBSTREAM AND PROGRAM SETUP	3
III	INTERFACE BETWEEN ROUTINES	13
IV	EAGLE BASICS	16
V	APPLICATIONS	68

LIST OF FIGURES

Figure	Title	Page
1	Points, Segments, and Spacings	20
2	Segments for Curves or Surfaces	21
3	Numbering a Body of Revolution	22
4	GETEND Operation	22
5	GETEND Using "FIRST" and "LAST"	23
6	SETNUM Operation with MATH = SUM-1	24
7	SETNUM Operation with MATH = DIF+1	25
8	Dimensions of Surface Segment	26
9	Points Specification Along a Line	29
10	Dimensions of a Surface	29
11	Specification of End Spacing	31
12	Construction of a Space Curve	33
13	Angle and Direction Definitions	34
14	Space Curve Using INSERT	35
15	Insertion of a Segment	36
16	Extraction of a Portion of a Surface	36

LIST OF FIGURES (continued)

Figure	Title	Page
17	Coordinate Unit Vectors	37
18	Surface Generation	40
19	Rotation Using Bounding Curves	40
20	Surface Generation bv Stacking	42
21	Surface Generation Using Circle and Ellipse	42
22	Example of a Stack to Generate a Surface	43
23	Surface Generation Using BLEND	44
24	Surface Generation by Transfinite Interpolation	45
25	INTSEC Operation	46
26	Intersection Curve to Generate a Surface	48
27	Example of Use of Intersection Curve	49
28	Curvilinear Coordinates	51
29	Curvilinear Coordinates Across Block Boundaries	52
30	Segment Read	54
31	Point Locations	57
32	Indirect Point Referencing	58

LIST OF FIGURES (continued)

Figure	Title	Page
33	Segment Specification	59
34	Griding an Existing Surface	67
35	Physical Space and Computational Space	69
36	'SWITCH' for Bottom Half of Airfoil	70
37	'INSERT' Merging Top and Bottom Half of Airfoil	71
38	Drawing the Half Circle for the Front Outer Boundary	71
39	Detail of Spacing for Front Outer Boundary	72
40	'SWITCH' for Points on Bottom Outer Boundary	72
41	Top Outer Boundary	73
42	Combining Segments to Form Complete Outer Boundary	73
43	'SWITCH' to Get Points Oriented Correctly on the Wake Line	73
44	Flow Field in Physical Space	79
45	Computational Space (Block 1 and Block 2)	80
46	Curve Describing Outer Surface	81
47	Curve of Outer Surface Transformed to Global Coordinates	81

LIST OF FIGURES (continued)

Figure	Title	Page
48	Boundary of Outer Surface of Solution Region	82
49	Stagnation Line Definition	83
50	Block 1, Stagnation Surface	84
51	Block 2, Stagnation Surface	84
52	Exit-Plane Definition	85
53	Block 1, Exit Boundary	86
54	Block 2, Exit Boundary	86
55	Line Segment 1 of Vehicle Boundary	87
56	Line Segment 2 of Vehicle Boundary	88
57	Insertion of Line Segment 2 to Segment 1	88
58	Point Redistribution of Line Segment 3	89
59	Line Segment 4 of Vehicle Boundary	90
60	Insertion of Line Segment 3 to End of Line Segment 4	90
61	Block 1, Vehicle Boundary	91
62	Block 2, Boundary of Outer Surface of Solution Region	92

LIST OF FIGURES (continued)

Figure	Title	Page
63	Block 2, Vehicle Boundary	93
64	Connectivity of Block 1 to Block 2	97
65	Files Created by Boundary Program and Their Location	98
66	Block to Block Interfaces by Cuts	99
67	Description of Flow Region Generated by Grid Program	101
68	Ogive-Cylinder-Ogive with Fins	104
69	Ogive-Cylinder-Ogive Boundary with Sting	108
70	46 x 2 Point Fin Tip Surface	109
71	Fin Surface	113
72	Missile Body Surface	115
73	Outer Boundary Outline	120
74	Outer Boundary Surface	122
75	Back Boundary Surface	123
76	Single Block Format	139
77	Single Block Format (Exploded View)	142

LIST OF FIGURES (concluded)

Figure	Title	Page
78	Four Block Layout	148
79	Front View (4 Block Grid)	149
80	Side View (4 Block Grid)	150
81	Three Finned-Body Configuration	155
82	30-Block Grid Scheme	156
83	Elliptic Grid Cross Section, Frontal View	157
84	Algebraic, Elliptic Grid Cross-Section, Side View	158
85	Perspective View of Wing-Pylon-Store, C-O Grid	161
86	Wing with Pylon, Store, and Sting	162
87	Initial Grid Around Wing, Pylon, and Store (Elliptic)	163
88	Separated Grid Around Wing, Pylon, and Store (Algebraic)	164

LIST OF TABLES

Table	Title	Page
1	Program EAGLE - Adjustable Dimension Parameters	9

SECTION I

INTRODUCTION

Program EAGLE (Eglin Arbitrary Geometry ImpLicit Euler) is a multiblock grid generation and steady-state flow solver system. This system combines a boundary conforming surface generation scheme, a composite block structure grid generation scheme and a multiblock, implicit Euler flow solver algorithm. The three codes are intended to be used sequentially from the definition of the configuration under study to the flow solution about the configuration. EAGLE has been specifically designed to aid in the analysis of both freestream and interference flowfield configurations.

These configurations can be comprised of single or multiple bodies ranging from simple axisymmetric airframes to complex aircraft shapes with external weapons. Each body can be arbitrarily shaped with or without multiple lifting surfaces.

Program EAGLE is written to compile and execute efficiently on any Cray machine with or without Solid State Disk (SSD) devices. Also, the code uses namelist inputs which are supported by all Cray machines using the Fortran Compiler CFT77. The use of namelist inputs makes it easier for the user to understand the inputs and to operate Program EAGLE.

Documentation for Program EAGLE is set up as follows:

Volume I: Introduction and Applications

Volume II: Surface Generation Routine

Volume III: Grid Generation Routine

Volume IV: Flow Solver Routine

Volume I is an introduction to the basics of EAGLE and its use, including several examples of weapon airframe and multiple body configurations.

Volume II deals solely with the theory and operation of the surface generation routine.

Volume III describes the theory and operation of the grid generation routine.

Volume IV covers the theory and operation of the Euler flow solver and presents sample surface pressure distributions of some of the applications in Volume I.

Program EAGLE has been a joint development effort between the Air Force Armament Laboratory's (AFATL) Aerodynamics Branch (FXA) and Mississippi State University's (MSU) Department of Aerospace Engineering.

SECTION II

JOBSTREAM AND PROGRAM SETUP

As mentioned earlier, Program EAGLE has been developed for Cray systems and is optimized for Cray X-MP machines using Solid State Disk (SSD) devices. Also, Program EAGLE has been written for use on Cray-2 machines. Both versions are in use and available through the Defense Technical Information Center (DTIC).

This section will concentrate on describing the set up of the jobstream for both the Cray X-MP and Cray-2, along with a full description of the various dimensions that can be updated for particular problems to conserve storage. The user is advised to review the installation manuals covering the Job Control Language (JCL) for the particular system used.

SURFACE GENERATION ROUTINE

A typical jobstream for the surface generation routine is listed below. The sample jobstreams reflect use on the CDC/SIS Cray X-MP, but the sequency of operations is similar on other machines.

```
 1)  JOB, T(jobtime), L (job priority).
 2)  ACCOUNT, (USERNAME), (PASSWORD), (PROJECT).
 3)  GET, SOURCE = (SURFACE CODE)/CI = TTY.
 4)  CFT, I = SOURCE, B = BINSURF.
 5)  SEGLDR, GO, CMD = 'BIN = BINSURF'
 6)  PUT, FT50 = SURFILE.
 7)  PUT, FT51 = PLOT/CO = TTY.
 8)  DFD, DAYFILE
 9)  PUT, DAYFILE/CO = TTY.
10)  PUT, $OUT = OUTFILE/CO = TTY.
11)  EOR
12)  E$INPUT ITEM = 'POINT', POINT = 1, R = 0.0, 0.0, 0.0 $
                    .
                    .
```

```
                    .
                Surface inputs
                    .
                    .
                    .
n)  E$INPUT ITEM = 'END'$
```

The first two lines are the generic job and account statements normally required. Line 3 gets the surface generation routine and compiles it in line 4, saving the compiled file as BINSURF. Line 5 loads and executes BINSURF. Line 6 saves the file required by the grid generation routine as SURFILE. Line 7 saves a formatted file for plotting of a specific segment or segments of the generated surfaces (see Section 4). Lines 8-10 save the output and dayfiles for debugging purposes. Line 12 begins the namelist inputs.

GRID GENERATION ROUTINE

A typical jobstream for the grid generation routine is listed below.

```
 1)  JOB, T(job time), L(job priority), SSDXXXXX.
 2)  ACCOUNT, (USERNAME), (PASSWORD), (PROJECT).
 3)  GET, SURFILE.
 4)  ASSIGN, DN = SURFILE, A = FT11.
 5)  GET, SOURCE = (GRIDCODE)/CI = TTY.
 6)  ASSIGN, DN = SSD07, A = FT07, DV = SSD, NOF.
 7)  ASSIGN, DN = SSD09, A = FT09, DV = SSD, NOF.
 8)  ASSIGN, DN = SSD10, A = FT10, DV = SSD, NOF.
 9)  ASSIGN, DN = SSD20, A = FT20, DV = SSD, NOF.
10)  ASSIGN, DN = SSD31, A = FT31, DV = SSD, NOF.
                    .
                    .
11)  ASSIGN, DN = SSD (NB+30), A = F(NB+30), DV = SSD, NOF.
12)  CFT, I = SOURCE, B = BINGRID.
13)  SEGLDR, GO, CMD = 'BIN = BINGRID'.
14)  LJT, $OUT = OUTFILE/CO = TTY.
15)  PUT, FT08 = PLTGRID/CO = TTY.
16)  PUT, FT12 = GRID.
```

```
17)  DFD, DAYFILE.
18)  PUT, DAYFILE/CO = TTY.
19)  EOR
20)  E$INPUT ITEM = 'STORE', FILE = 12, ITMAX = 50$
                          .
                          .
                          .
                    GRID INPUTS
                          .
                          .
                          .
21)  E$INPUT ITEM = 'END'$
22)  E$OUTPUT ITEM = 'ERROR'$
                          .
                          .
                          .
                   GRID OUTPUTS
                          .
                          .
                          .
23)  E$OUTPUT ITEM = 'END'$
```

The first two lines are as before with the exception of the SSD parameter.

When using Solid State Disk (SSD), the JOB statement requires an additional parameter. For the JOB statement, the user must add "SSDXXXXX" where XXXXX is the total number of blocks of SSD required. Each block is equivalent to 512 words, and the number of blocks specified should be divisible by 32. The third line gets the file that was saved in the surface code containing the boundary surfaces. Line 4 assigns this file to FT11. Line 5 gets the grid generation routine. Lines 6 through 9 assign four SSD files for internal use by the code. Line 10 begins a series of ASSIGN statements; one statement for each block in the grid. These SSD files are labeled as "30 plus the block number" with an example of the last block, NB, statement as line 11. When using SSD, the ASSIGN statement requires an additional parameter. The ASSIGN statement requires the device type to be specified as "DV=SSD" for each file that will use SSD. In other words, when

working a multiblock problem, the parameter "DV=SSD", in the ASSIGN statement, must be included to specify the file"30+XX" as an SSD file (where XX is a user supplied file number with certain restrictions mentioned in Volume II). One optional parameter that is of importance is the NOF term (NO OVERFLOW). This term assures that if the SSD limit in the JOB statement is exceeded while using the particular SSD file, program execution will stop. Without this, the program will continue by writing to conventional disk. This can become extremely expensive for large problems where conventional disk memory is accessed many times. Of course, other parameters are available and will not be discussed here. The user should refer to the installation manuals for more detailed information.

Line 12 compiles the source code and saves the compiled version as BINGRID. Line 13 then loads and executes BINGRID. Line 14 saves the output file for debugging while a formatted file for plotting specific segments of the grid is saved in line 15. Line 16 saves the actual grid to be used as an input to the flow solver. Lines 17 and 18 establish and save the day file record of the job. Line 20 begins the grid inputs. Note the file number to store the grid (12) is the same as that in line 16. Line 22 begins the grid outputs which are useful in debugging and tracking the convergence of the grid solution.

FLOW SOLVER ROUTINE

A typical jobstream for the flow solver routine is listed below.

```
 1)  JOB, T(jobtime), L(job priority), SSDXXXXX.
 2)  ACCOUNT, (username), (password), (project)
 3)  GET, GRID.
 4)  ASSIGN, DN = GRID, A = FT10.
 5)  GET, = (FLOWCODE)/CI = TTY.
 6)  CFT, I = (FLOWCODE), B = BINFLOW.
 7)  SEGLOR, GO, CMD= 'BIN = BINFLOW'.
 8)  PUT, FT01 = (RESTART).
 9)  PUT, $OUT = OUTFILE/CO = TTY.
10)  DFD, DAYFILE.
11)  PUT, DAYFILE/CO = TTY.
```

```
12)  EOR
13)  E&FINPUT CFL = 5.0, FSMACH = 0.95, NP = 1, NT = 1000$
                              .
                              .
                              .
                         EULER INPUTS
                              .
                              .
                              .
14)  E$BCIN IBTYPE =  0$
```

Lines 1 and 2 are the same as those for the grid generation routine. Line 3 gets the grid file and line 4 assigns it to FT10. The flow code is obtained in line 5, compiled in line 6 and loaded and executed in line 7. Line 8 saves a restart file to be used if additional iterations of the flow code are required later. Lines 9-11 save the output and dayfiles. Line 13 begins the flow code inputs.

ADJUSTABLE DIMENSIONS

Program EAGLE has several dimensions that can be changed to meet particular job requirements (or to keep storage to a minimum) by editing the applicable code and making global changes on specific parameters. These dimensions can be broken down into three sections, one applicable to each code. These dimensional variables with their preset values are shown in Table 1.

SURFACE

The variables DIM1 and DIM2 are the maximum number of points that can describe a curve (DIM1) or a surface (DIM1 x DIM2). In other words, when developing a curve with N1 points or a surface with N1 x N2 points, then

(for a curve) $N1 \leq DIM1$

(for a surface) $N1 \leq DIM1$ and $N2 \leq DIM2$

The variables DFIL and DCOR are the maximum number of FILEOUTs and FILEINs and COREOUTs and COREINs, respectively, allowed within the surface generation input stream (see Volume II, Section B.2).

DIMV is the maximum number of values that can be read in Namelist. This number cannot be greater than DIM1 for a curve, or DIM1 x DIM2 for a surface (see Volume II, Section I-B1).

The variables NVALMX and DVAL are used when storing a sum, difference, or a product. The maximum number of values in the calculation cannot exceed NVALMX, while the maximum number of in-core storage locations cannot exceed DVAL (see Volume II, Section I-B1).

TABLE 1. PROGRAM EAGLE - ADJUSTABLE DIMENSION PARAMETERS

Routines	Dimension Variables	Preset Values
Surface	DIM1	501
	DIM2	110
	DFIL	100
	DCOR	500
	DIMV	2000
	DPNT	500
	NVALMX	100
	DVAL	500
	DIMSS	10000
Grid	NS	1
	BASE	30
	DSUB	20
	DIMB	48
	DIMD	50000
	DPNT	1000
	DSEG	1000
	NVALMX	100
	DVAL	500
	DIMC	2000
	DIMT	35000
	DMRB	10000
	DIMP	100
	DIMS	10000
	DIML	170
	DNEU	6
	DREF	6
	DORT	6
	DIMI	10000
	DMNT	1
	DMOT	1000
	DIMR	1
	DIMN	1
	DIMO	1
	DIMG	1
Flow	MAXSURF	20
	MAXBLK	24
	MAXBC	500
	S	1 (for Cray X-MP)
	S	100,000,000 (for Cray-2)

The maximum number of points that can be placed in core storage before automatically writing out to a file is given by DIMSS. This value is difficult to determine; however, the absolute maximum can be estimated to be

$$DIMSS \geq 6 \times BMAX \times (DIM1 \times DIM2)$$

where BMAX is the maximum number of blocks (see Volume II, Section I-B1). This, of course, will require large amounts of in-core memory. The user will have to establish a desirable medium between the amount of in-core storage used and the amount written to a file.

Finally, DPNT is the maximum number of numbered points that can be used (see Volume II, Section I-B1).

GRID

The grid code prints a summary of resources actually used to guide the user in considerations of whether to change the adjustable dimension to reduce storage.

The variable NS gives the number of surrounding layers for each block.

The variable BASE indicates the base file which will be used as scratch files for the individual block grids starting with BASE+1 to BASE+BMAX. Note that files 1 through 10 and 20 cannot be used. These are reserved as scratch files for code usage.

The maximum number of sub-blocks is given by DSUB, while the maximum number of blocks is given by DIMB (see Volume III, Section II-A2).

The maximum number of points in a block is given by DIMT. This variable can be determined by the following equation:

$$DIMT \geq (N1 + 2 \times NS) \times (N2 + 2 \times NS) \times (N3 + 2 \times NS)$$

and must satisfy the condition:

$$15 \times DIMT \geq 12 \times DIML^2$$

where:

N1, N2, N3 - points in the three directions of a block, and

DIML - is the maximum number of points on a block edge.

The variable DMRB also is the maximum number of points on a block boundary. It is determined from:

$$DMRB \geq 8 + 2 \times [(N1+NS)\times(N2+NS) + (N2+NS)\times(N3+NS) + (N3+NS)\times(N1+NS)] -$$

$$4 \times [(N1+NS) + (N2+NS) + (N3+NS)]$$

The variable DIMP specifies the maximum number of points that can be read in from the namelist at one time.

The maximum number of points on a block side is given by the variable DIMS and can be determined from the following equation:

$$DIMS \geq MAX(N1\times N2,\ N2\times N3,\ N3\times N1)$$

The maximum number of points on a block edge is given by the variable DIML. DIML can be determined from the following equation:

$$DIML \geq MAX(N1,\ N2,\ N3)$$

The variables DNEU, DORT, and DREF specify the maximum number of Neumann, orthogonal, and reflective boundary sections, respectively (see Volume III, Section II-A2).

The variables DIMI, DMNT, and DMOT specify the maximum number of image, Neumann, and orthogonal boundary points in a block, respectively. These variables cannot be determined explicitly. The maximum would be, of course, DIMT; however, this would be overestimating the maximum. The user will have to determine these three values.

The variable, DIMR, is the total number of points in all the blocks, while the other three variables, DIMN, DIMO, and DIMG are the total number of Neumann, orthogonal, and image points in all the blocks, respectively. These four variables are irrelevant and are set to one when file space, rather than in-core memory space, is used (i.e., when KSTORE = "FILE", see Volume III, Section II-A2). Otherwise, when KSTORE = "CORE", these variables set up the

arrays for in-core memory usage only. DIMR can be determined by the following equation:

$$DIMR \geq \sum_{i+1}^{DMAX} [N1_i+2xNS) \times (N2_i+2xNS) \times (N3_i+2xNS)]$$

The other three variables cannot be determined explicity. The user must estimate these values.

The parameter DIMD is the maximum number of boundary points that can be included on one combined read of data filed by the COMBINE option of the surface code (see Volume II, Section I-B1 and Volume III, Section II-A2). Similarly, DSEG is the maximum number of boundary segments that can be read from such a file.

The parameter DIMC is the maximum number of points allowed on a curved surface on which a grid is to be generated. This parameter can be set to 1 for 3D operation (and for 2D operation on a plane).

The parameters DPNT, NVALMX, and DVAL have the same meaning given above for the surface code.

FLOW

The variable MAXSURF limits the number of surfaces on which force and moment values and coefficients are calculated. One surface could be as large as the entire configuration or as small as one cell.

MAXBLK limits the number of grid blocks that can be used.

MAXBC limits the number of boundary conditions that can be read in. Each BCIN input line with IBTYPE equal 1,2,3 or 6 count as one boundary condition while those equal to 4 or 5 count as two. The sum for all BCIN lines must not exceed MAXBC.

The ribbon vector S contains all the variables calculated within the code for all blocks. On the Cray X-MP where SSD files are opened and closed inside the code, transparent to the user, the value is set to 1 and need not be changed. On the Cray-2, all the variables are stacked end to end for each block and is subject to the limitation on S. For most problems, the preset value is sufficient but it can be raised for large problems or lowered to save core storage costs.

SECTION III

INTERFACE BETWEEN ROUTINES

The generation of the six computational surfaces (four boundaries, in 2-D) of the blocks in the SURFACE GENERATION routine feeds into the GRID GENERATION routine to generate a 3-D field grid. The formats of these outputs from the SURFACE or GRID routines may be of importance to those who may wish to replace any of the three primary routines or add any of their own routines.

As discussed previously, the SURFACE routine outputs the coordinate files to be used by the GRID routine as inputs to develop the complete field grid. These output coordinate files are written, unformatted, to a user specified or files using a COMBINE statement or individual FILEOUT statement file (see Volume 2 for details) from a three-dimensional array storing the coordinates. The format of this write statement is as follows:

```
      DO     20    C2 = 1, N2
      DO     20    C1 = 1, N1

20    WRITE (10+XX) (F(CI, C1, C2), CI = 1, 3)
```

Where:

C1 and N1 - number of points in 1st direction (faster running direction)

C2 and N2 - number of points in 2nd direction

CI - Cartesian components

CI = 1 -> X

CI = 2 -> Y

CI = 3 -> Z

F - three-dimensional array storing the Cartesian coordinates of the surfaces (or curves in 2-D) (REAL)

The GRID routine generates an unformatted output file dependent on the user selection for the parameter "OUTER" in the grid input list (see Volume

III for details). Four options exist for "OUTER" - "YES", "NO", "BOUND", and "SEPARATE".

If OUTER = "YES" then subroutine WRTGRD outputs the classification of the points, the surrounding layer of points, and the Cartesian coordinates for each block on file "GRIDFIL". The format for these write statements is:

```
      Write (GRIDFIL) BMAX, ((CMAX (CI,B), CI = 1,3), B = 1, BMAX

      DO 3001 B = 1, BMAX
      DO 3001 C3 = IS, CMAX (3,B) + NS
      DO 3001 C2 = IS, CMAX (2,B) + NS
      DO 3001 C1 = IS, CMAX (1,B) + NS

      Write (GRIDFIL) Type (B, C1, C2, C3)

      Write (GRIDFIL) (Image CI, B, C1, C2, C3), CI = 0, 3)

3001  Write (GRIDFIL) (R(CI, B, C1, C2, C3), CI = 1,3)
```

Where: *BMAX - Maximum number of blocks (Integer)*

IS - Integer value specified in parameter statement

NS - Integer value specified in parameter statement (see Section II for definition)

CMAX - Two-dimensional array containing the maximum number of points in each direction for every block (Integer)

TYPE - Four-dimensional array containing the classification of all the points for each block

IMAGE - Five-dimensional array containing the surrounding layer of points for each block (Real)

R - Three-dimensional array containing the Cartesian coordinates of every point for each block (Real)

If the user selects **OUTER = "NO"**, then Subroutine WRTRRR outputs the Cartesian coordinates only for each block on file "GRIDFIL". The format for these write statements is:

```
Write (GRIDFIL) BMAX, ((CMAX(CI,B), CI + 1,3), B = 1, BMAX)
DO 3002 B = 1, BMAX
DO 3002 C3 = 1, CMAX(3,B)
DO 3002 C2 = 1, CMAX(2,B)
DO 3002 C1 - 1, CMAX(1,B)

3002 Write (GRIDFIL) (R(CI, B, C1, C2, C3) CI = 1,3)
```

The option OUTER = "BOUND" operator as does "SEPARATE", but omits the first line containing BMAX and CMAX.

For the last option - **OUTER + "SEPARATE"** - Subroutine WRTXYZ will write the Cartesian coordinates in three separate arrays for each block on the files "BASE+B" where B is the block number. The format for these write statements is:

```
Write (GRIDFIL) BMAX, ((CMAX(CI, B), CI=1,3), B=1, BMAX)
DO 5000 C3 = 1, CMAX(3,B)
DO 5000 C2 = 1, CMAX(2,B)
DO 5000 C1 = 1, CMAX(1,B)
X(C1, C2, C3) = R(1,B, C1, C2, C3)
X(C1, C2, C3) = R(2,B,C1,C2,C3)
X(C1, C2, C3) = R(3,B,C1,C2,C3)
REWIND BASE+B
WRITE (BASE+B) X,Y,Z
```

Where: X,Y, and Z - Three-dimensional arrays containing the X, Y, and Z coordinates, respectively, for every point (Real)

The outputs of the four options can be used by any external flow code or plot code. The Program EAGLE - Flow Solver uses option 2.

SECTION IV

EAGLE BASICS

1. THE EAGLE CODE

The EAGLE grid code is a general three-dimensional elliptic grid generation system based on a composite block structure. This code allows any number of blocks to be used to fill an arbitrary three-dimensional region. Any block can be linked to any other block (or to itself), with complete (or lesser) continuity across the block interfaces as specified by input. In the case of complete continuity, the interface is a branch cut, and the code establishes a correspondence across the interface using a surrounding layer of points outside the blocks. This allows points on the interface to be treated just as all other points, so that there is no loss of continuity. The physical location of the interface is thus totally unspecified in this case, being determined by the code.

This code uses an elliptic generation system with automatic evaluation of control functions, either directly from the initial algebraic grid and then smoothed, or by interpolation from the boundary point distributions. In the former case the smoothing is done only in the two directions other than that of the control function. This allows the relative spacing of the algebraic grid to be retained but on a smoother grid from the elliptic system. In the latter case, the arc length and curvature contributions to the control functions are evaluated and interpolated separately into the field from the appropriate boundaries. The control function at each point in the field is then formed by combining the interpolated elements. This procedure allows very general regions, with widely varying boundary curvature, to be treated.

The control functions can also be determined automatically to provide orthogonality at boundaries with specified normal spacing. Here the iterative adjustments in the control functions are made by increments radiated into the field from boundary points where orthogonality has not yet been attained. This allows the basic control function structure evaluated from the algebraic grid, or from the boundary point distributions, to be retained and thus relieves the iterative process from the need to establish this basic geometric form of the control functions.

Alternatively, boundary orthogonality can be achieved through Neumann boundary conditions which allow the boundary points to move over a surface spline, the boundary point locations being located by Newton iteration on the spline to be either at the foot of normals to the adjacent field points or on extrapolated straight lines from the two adjacent field points. Provision is also made for mirror-image reflective boundary conditions on symmetry planes.

Although written for 3D, the code can operate in a 2D mode on either a plane or curved surface. In the case of a curved surface, the surface is splined and the generation is done in terms of surface parametric coordinates.

The code includes an algebraic three-dimensional generation system based on transfinite interpolation (using either Lagrange or Hermite interpolation) for the generation of an initial grid to start the iterative solution of the elliptic generation system. This feature also allows the code to be run as an algebraic generation system if desired, taking this initial algebraic grid as the final product. The interpolation, though defaulted to complete transfinite interpolation from all boundaries, can be restricted by input to any combination of directions or lesser degrees of interpolation, and the form (Lagrange, Hermite, or incomplete Hermite) can be different in different directions or in different blocks. The blending functions can be linear or, more appropriately, based on interpolated arc length from the boundary point distributions.

Blocks can be divided into sub-blocks for the purpose of generation of the algebraic grid and the control functions. Here point distributions on the sides of the sub-blocks can either be specified or generated by transfinite interpolation from the edges of the side. This allows additional control over the grid in general configurations and is particularly useful in cases where point distributions need to be specified in the interior of a block, or to prevent grid overlap in highly curved regions. The code will automatically interpolate for any values on block interfaces, or on sub-block sides, that are not read in.

The composite structure is such that completely general configurations can be treated, the arrangement of the blocks being specified by input, without modification of the code. The input is user-oriented and designed for brevity and easy recognition. For example, the establishment of correspondence, i.e. a branch cut, between two blocks requires only the simple NAMELIST input statement,

$INPUT ITEM = "CUT", START = __,__,__, END = __,__,__, BLOCK = __,

ISTART = __,__,__, IEND = __,__,__, IBLOCK = __$,

where START and END give the three indices of two opposite corners of the cut section on one block (BLOCK), while ISTART and IEND give the corners of the corresponding section on the other block (IBLOCK). The code sets up the point correspondence on the surrounding layers for complete continuity without additional input instructions. A NAMELIST emulator is available for machines that do not support NAMELIST input.

The code does extensive error checking to assist the user in the input construction, and is designed to note any omissions on the input and not to stop without an explanation.

Detailed discussion of both the use and the operation of the code is given in Volumes II and III. This present section provides a quick introduction to the use of the code, covering the essential and most used features. Volumes II and III should be consulted for complete information on all features.

The code is written in modular form so that components can be readily replaced. The code is vectorized wherever practical and includes provision for separate storage of each block on the CRAY solid-state disk (or conventional disk) to allow very large grids to be generated. The various blocks can thus either be kept on disk file, with only a single block in core at one time (e.g. CRAY X-MP), or can all be kept in core without using disk storage (e.g. CRAY 2).

2. BOUNDARY CODE

The boundary code (Volume II) builds up the boundary segments for input to the grid code (Volume III). This is done by a series of operations which read in data, create certain geometric forms, perform geometric transformations, and make combinations.

Each of these operations is invoked by a NAMELIST input statement (called an "operation statement" here) of the form

$INPUT ITEM = "__", --- $,

with the alphanumeric name of the particular operation within the quotes, and the relevant quantities for that operation included on the operation statement as described below. The result (a curve or surface segment) of each such operation is automatically in position to be acted on by the next operation. The resulting segment can also be assigned a unique segment number and stored for later access by including

COREOUT = segment number

on the operation statement. This segment can then be accessed by including

COREIN = segment number

on a later operation statement.

The segment numbers for completed boundary segments can be carried over from the boundary code into the grid code and used to address these segments there as well. However, it is not necessary that the boundary code be coupled with the grid code in this manner, for the grid code can operate independently, accepting boundary segments from any source. This coupling does, however, allow changes in the number of points, or other factors, on boundary segments to be made locally in one place in the input runstream for the boundary code without requiring corresponding changes in the runstream for the grid code.

a. Indirect Addressing

In the construction of the boundary segments in the boundary code, it will be necessary to specify certain points in space, such as the end points of lines and curves, etc., on the operation statements. Such a point can be indicated directly by supplying its three Cartesian coordinates, or indirectly by an assigned point number to which the three Cartesian coordinates have been attached by another operation statement for that purpose. Similarly, the number of points to be placed on a curve can be either given directly or can be attached to the segment number assigned to the curve. Spacings can also be assigned numbers by which they can be identified whenever used. These indirect identification features are not required, and all quantities can be given directly wherever used if desired. However, since many quantities are used more than once, these features do allow changes to be made locally in one place in the runstream without corresponding changes being required throughout the runstream.

If this indirect addressing of quantities is used, there will typically be point numbers, segment numbers, and spacing numbers (indicated by numbers in circles, squares, and triangles, respectively, in the following diagram):

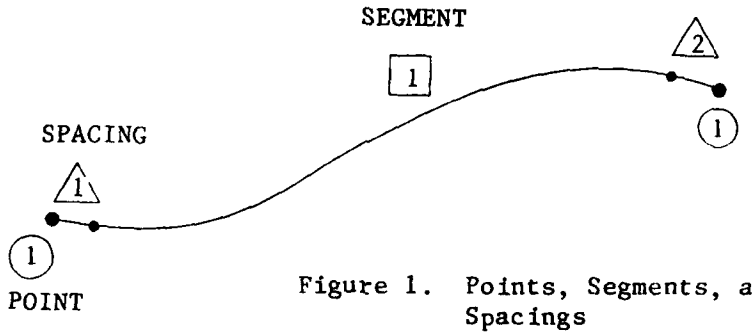


Figure 1. Points, Segments, and Spacings

Thus, curve segment #1 here connects points #1 and #2, and the points adjacent to the ends of the curve are to be placed at spacings #1 and #2. No relation is implied among the point, segment, or spacing num-

bers, i.e., neither segment #1 nor spacing #1 need be associated with point #1. There may be surface segments, as well as curve segments, as shown below:

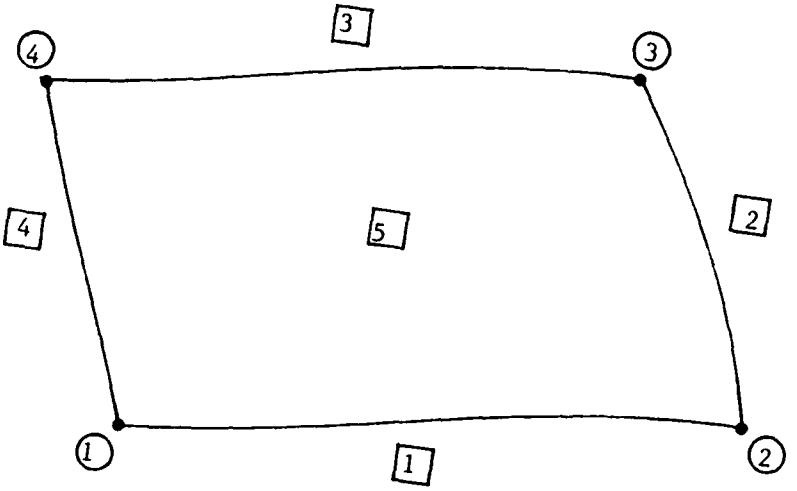


Figure 2. Segments for Curves or Surfaces

Here surface segment #5 is bordered by the four curve segments #1,2,3,4, which are terminated by the points #1,2,3,4.

No distinction is made between curve segments and surface segments in regard to numbering, so the same number cannot be used for both a curve and surface segment. Point, segment, and spacing numbers are unrelated, however, so that the same number can be used for one of each as in the above illustrations. Again, no relation between the point and segment numbers is implied, and it is not necessary that the numbers of either be consecutive, or that all numbers be used, i.e., gaps can be left. It is, in fact, often helpful to use groups of numbers, e.g. 100's, 200's, etc. for certain natural groupings of points or segments. For example, a group of points on a 0° meridional plane for a body of revolution might be numbered 1-10, while the corresponding points on the 90°, 180°, and 270° planes are numbered 101-110, 201-210, and 301-310, respectively:

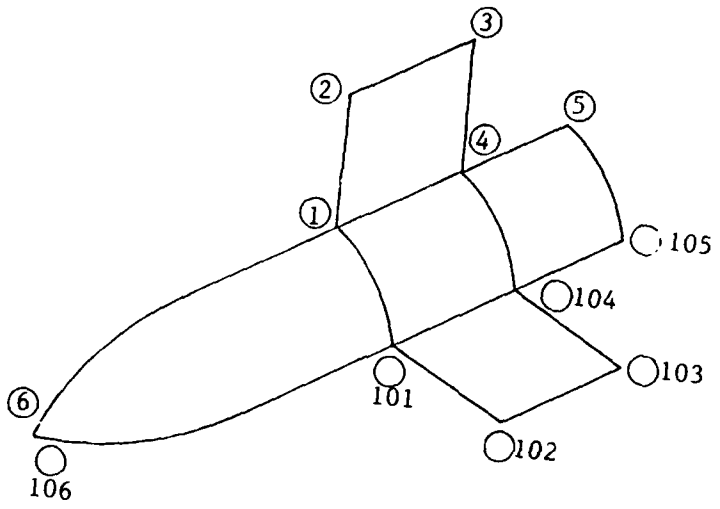


Figure 3. Numbering a Body of Revolution

Point Numbers

In this indirect addressing mode, the three Cartesian coordinates can be attached to a point number in two ways:

(1) directly, through the operation statement

$INPUT ITEM = "POINT", POINT = point number, R = x,y,z $

(2) from a point on an existing segment, through the two consecutive operation statements

$INPUT ITEM = "GETEND", COREIN = segment number, POINT = i,j $

$INPUT ITEM = "POINT", POINT = point number $

Here (i,j) are the indices of the point on the segment. (This segment must have been generated by some previous operation and stored there by an included COREOUT=segment number.)

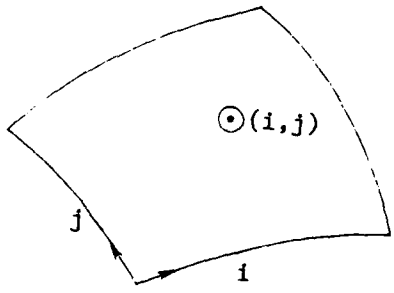


Figure 4. GETEND Operation

These two consecutive operations attach the Cartesian coordinates of the point at (i,j) on the existing segment indicated by COREIN on the "GETEND" statement to the point number indicated by POINT on the "POINT" statement. This point can then be used in later operations. If the existing segment is a curve, then POINT=i,j on the "GETEND" statement is replaced by POINT=i, and if the point intended is one of the end points of this curve, even by POINT="FIRST" or "LAST", indicating the first or last end point, respectively.

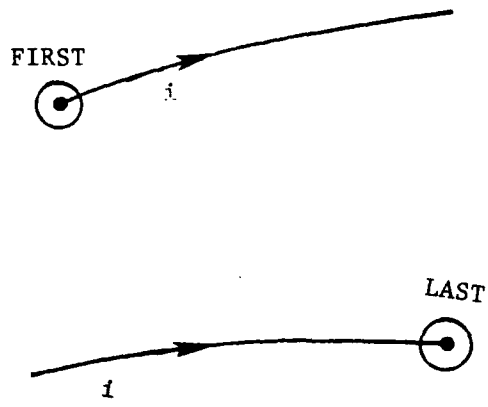


Figure 5. GETEND Using "FIRST" and "LAST"

With this indirect addressing, the point number (a single positive integer) can then be given in place of the three Cartesian coordinates on all operation statements requiring the coordinates of that point. It is the presence of a single number, rather than three, that keys the code to obtain the corresponding three Cartesian coordinates previously assigned to that point from storage. This assignment must, of course, have been done by a prior operation statement with ITEM="POINT" as explained above.

Number of Points on a Curve Segment

The number of points to be on a curve segment can be set in two ways also:

(1) directly, through the operation statement

$INPUT ITEM = "SETNUM", SEGMENT = segment number,
POINTS = number of points $

(2) as a sum or difference of the number of points on other segments, through the operation statement

$INPUT ITEM = "SETNUM", SEGMENT = segment number,
ITERMS = -other segment number, -other segment number, ---,
MATH = "__" $

Here the number of points on each of the other segments included in ITERMS must already have been set by previous "SETNUM" operations. If MATH="SUM-1", the number of points on the segment will be calculated as the sum of the number of points on all of the other segments (any number of other segments) given for ITERMS. This sum is performed by adding the number of points, less one, so that the ends are not counted twice, on each succeeding segment in ITERMS:

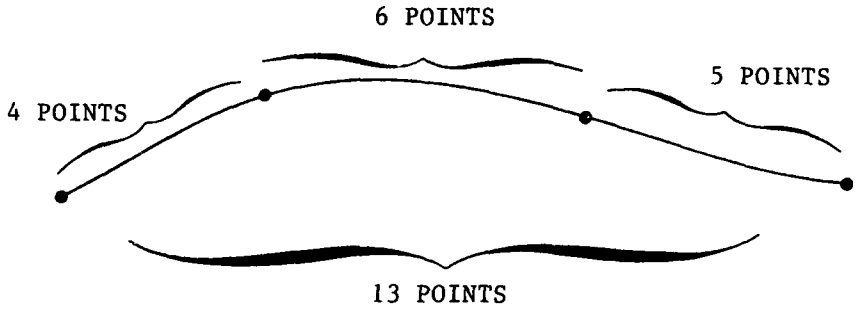


Figure 6. SETNUM Operation with MATH = SUM-1

(Since "SUM-1" is the default, MATH can actually be omitted from the statement in this case.) If MATH="DIF+1" the number of points is calculated by subtracting the number of points, less one, on the succeeding segments in ITERMS from that on the first:

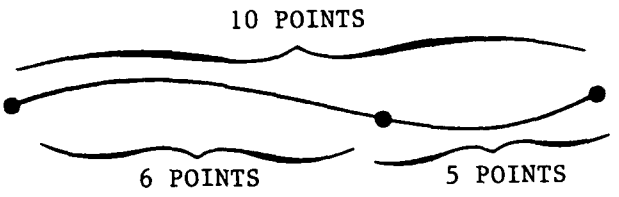


Figure 7. SETNUM Operation with MATH = DIF+1

There are actually some other possibilities for MATH, and Volume II should be consulted. The two given here are, however, by far the most common. Also, note that in the above description, each segment number in ITERMS is preceded by a minus sign. The omission of this minus sign causes the entry to be interpreted directly as a number of points, instead of a segment number, to be used in the calculation. ITERMS can, in fact, contain a mixture of both of these forms.

Although both curve and surface segments can be numbered, numbers of points can be set by "SETNUM" only on curve segments. The two dimensions of a surface segment can be set by numbering two of the curve segments forming the edges of the surface in two directions and using two operation statements with ITEM="SETNUM" to set the numbers of points on these two curve segments.

With this indirect addressing, the negative of the segment number can then be given in place of the number of points on all operation statements requiring the number of points on that segment. For a surface segment, for which the number of points in each of two directions must be specified, the indirect addressing can be used for either or both

directions. In this case the segment numbers used in the indirect addressing are those of curve segments forming edges of the surface in two directions, not the surface segment number itself:

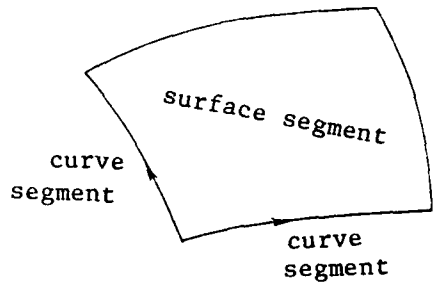


Figure 8. Dimensions of Surface Segment

It is the presence of the minus sign preceding the integer segment number that keys the code to recognize the integer as a segment number and thus to obtain the corresponding number of points assigned to that segment from storage. This assignment must, of course, have been done by a prior operation statement with ITEM="SETNUM" as explained above. Once the number of points on a curve segment is specified by ITEM="SETNUM", it can be changed only through a subsequent SETNUM operation.

Spacings

Finally, spacings can be assigned spacing numbers in two ways:

(1) directly, through the statement

```
$INPUT ITEM = "SETVAL", NUMBER = spacing number,
       VALUE = spacing $
```

(2) as a sum, difference, or product of other spacings through the statement

```
$INPUT ITEM = "SETVAL", NUMBER = spacing number,
      TERMS = -other spacing number,  -other spacing number, ---,
       MATH = "__" $
```

Here the spacings indicated by the other spacing numbers (integer) in TERMS must already have been given values by previous "SETVAL" operations. There are three possible types of calculations, indicated by "SUM", "DIF", or "PRODUCT" for MATH. The sum and product are performed for all entries in TERMS (any number), while the difference is subtraction of the second entry from the first. The minus sign preceding the spacing numbers in TERMS indicates that these are spacing numbers of spacings to be used in the calculation. The omission of this sign causes the number given (now a real number) to be taken as a value itself to be used in the calculation. It is possible to mix spacing numbers and values in TERMS.

With this indirect addressing, the negative of the spacing number can then be given in place of the spacing on all operation statements requiring that spacing. It is the presence of a negative integer, instead of a positive number, that keys the code to obtain the spacing assigned to that number from storage. This assignment must, of course, have been done by a prior operation statement with ITEM="SETVAL" as explained above.

The same identification number cannot be used for more than one spacing, although these spacing numbers are unrelated to point and segment numbers.

Settings

Although it is logical to group the operation statements that set the coordinates points ("POINT"), the number of points on segments ("SETNUM"), and the value of spacings ("SETVAL"), at the beginning of the runstream for the boundary code, such statements can be scattered throughout the runstream if desired. The only requirement is that the setting of a quantity be done before this indirect addressing is used for that quantity on other operation statements. These quantities can, in fact, be reset at different points in the runstream if it is useful to do so.

b. Segment Construction

The indirect addressing mode just discussed is an optional feature of the boundary code. The operations of this code build up the boundary segments by constructing, combining, and manipulating surface and curve segments. The coordinates of curve end points, the number of points on segments, and the values of spacings involved in these operations can be set using this indirect addressing or can be set directly on each operation statement. The formation of a boundary segment for input to the grid code typically involves a sequence of these operations, each of which may use the result of the previous operation or some other preceding operation. Defaults are reset after each operation.

Reading Segments

The most basic operation is, of course, simply to read in the Cartesian coordinates of a set of points which constitute a curve or surface segment. This is done by the operation statement

```
$INPUT ITEM = "CURRENT",
POINTS = dimensions of curve or surface,
VALUES = x, y, z,
         x, y, z,
            .
            .
            .
         x, y, z, COREOUT = segment number $
```

The inclusion of COREOUT, which is optional, assigns a segment number to the curve or surface as read in, as noted above, by which the segment can be accessed by later operation statements. If the set of points being read in constitutes a curve, then the total number of points being read is indicated by a single integer given for POINTS,

Figure 9. Points Specification Along a Line

while for a surface two integers are given. In the latter case the first of the two integers indicates the number of points in the faster running direction (#1 direction) on the surface when stored on a file:

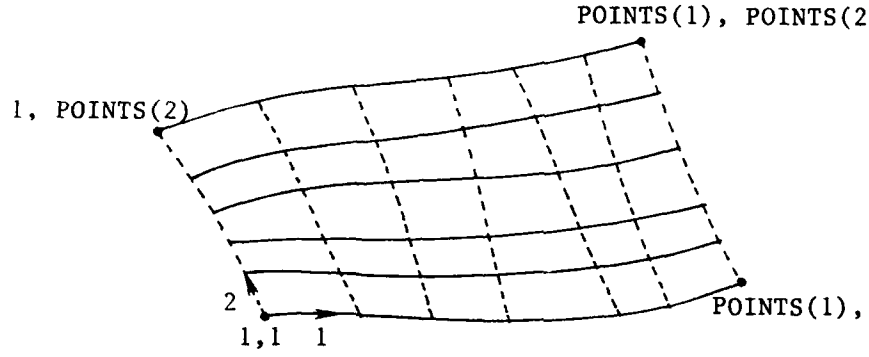


Figure 10. Dimensions of a Surface

The total number of points read for a surface is thus the product of the two dimensions indicated by the two entries of POINTS. A positive integer for either entry of POINTS is taken directly as the number of points. The indirect addressing mode discussed above can, however, be used for either or both entries of POINTS, i.e., a negative integer indicates a segment number to which a number of points has previously been attached by an ITEM="SETNUM" statement. This segment number does not have to be the same as that given here for COREOUT.

The Cartesian coordinates are given as triads of real numbers for each point in succession. (It is also possible to give only a pair of numbers x,y for each point, in which case z will be defaulted to 0.)

These coordinate values do not have to be arranged in columns as shown, but are given simply as a series of numbers separated by commas in any form.

It is also possible to read in the coordinate values from an external file by including FILE=filenumber, instead of VALUES, on the operation statement given above. In this case the form of the file can be either all three coordinates of a point on a single line (the default), or one coordinate to a line (indicated by including TRIAD="NO" on the operation statement). The format can be list-directed (indicated by including FORM="LIST" on the operation statement), formatted in E20.8 format (indicated by including FORM="E"), or may be unformatted (the default).

Point Distributions

After a curve has been read in as described above (or generated in any manner by other operations described later), any number of points can be distributed on the curve by a cubic spline with specified spacing on one or both ends by the operation statement

```
$INPUT ITEM = "CURDIST", POINTS = number of points,
       SPACE = first end spacing, second end spacing,
     COREOUT = segment number $
```

If the curve was not input by the immediately preceding operation statement, then it must be gotten from storage by including COREIN. The number of points to be placed on the curve can be given directly (positive integer for POINTS) or indirectly (negative integer for POINTS), as has been discussed above. (This number of points has no relation to the number of points used previously to define the curve.)

The spacings can be given directly (positive real value) or indirectly (negative integer). The operation statement given here assumes that the spacings are relative spacings, i.e., fractions (0-1) of the total arc length on the curve. Absolute spacings can also be used by including RELATIV, also with two entries corresponding to those of SPACE. Here "NO" for an entry of RELATIV indicates that the correspond-

ing spacing given is absolute, while "YES" (the default) indicates relative spacing. The absolute spacing can also be specified to be the same as that on an end of a previously generated curve by giving a positive integer as an entry of SPACE, the integer being the segment number (set by COREOUT when the other curve was generated) of the other curve. (Although absolute spacings are involved in this case, RELATIV is not to be included.) In this case END, again with two entries, must also be included, with "FIRST" or "LAST" as an entry of END indicating that the corresponding spacing is to match that at the first or last end of the other curve. The default for END is END="LAST", "FIRST", corresponding to the common case illustrated below.

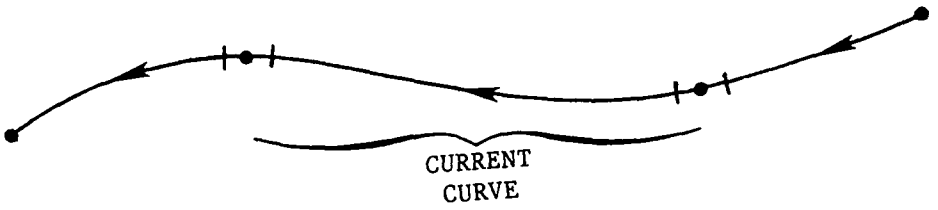


Figure 11. Specification of End Spacing

The curve from which the spacing is taken does not, however, have to be physically adjacent to one being treated. Finally, if only a single entry is given for SPACE, the spacing is set at only the first end of the curve. The omission of SPACE will produce an equally-spaced point distribution. (There are also some other options for which Volume II should be consulted directly.)

The resulting segment can be assigned a segment number and stored by including COREOUT as indicated, but this is not required.

Straight Line

A straight line between two points in space can be constructed by the operation statement

```
$INPUT ITEM = "LINE", POINTS = number of points,
       R1 = first point, R2 = last point,
     SPACE = first end spacing, last end spacing,
   COREOUT = segment number $
```

Here POINTS and SPACE can be given directly or indirectly as has been discussed above for ITEM="CURDIST". Either or both of the end points, R1 and R2, can be specified directly by giving the three Cartesian coordinates as three real numbers,

```
R1 = x,y,z  and/or  R2 = x,y,z
```

or can be specified indirectly by giving the point number as a single positive integer, as discussed above. The (optional) inclusion of COREOUT assigns a segment number to the line.

Cubic Curve

A cubic space curve between two points in space can be generated by the operation statement

```
$INPUT ITEM = "SCURVE", POINTS = number of points,
       R1 = first point, R2 = last point,
       T1 = first end slope, T2 = last end slope,
     SPACE = first end spacing, last end spacing,
   COREOUT = segment number $
```

Here all quantities serve as for ITEM="LINE", with the addition of the slope vectors, T1 and T2, each of which is given as three real numbers for the three direction cosines of the slope vector (i.e., the components of the unit slope vector).

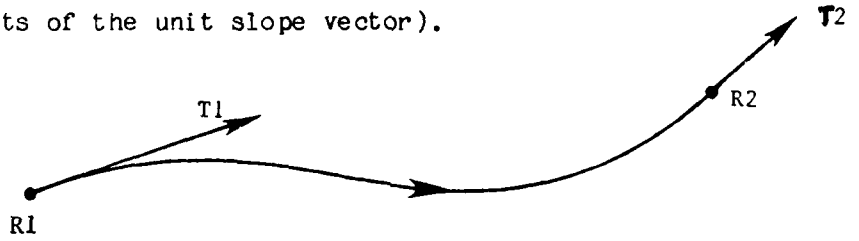


Figure 12. Construction of a Space Curve

Entries for T1 and T2 that are greater than unity are taken as actual angles (in degrees), of which the cosine is taken by the code. The following are thus equivalent:

T1 = 0.707, 0.707, 0.707 and T1 = 45,45,45

Circular Arc

A generic circular arc in the x-y plane can be generated by the operation statement

$INPUT ITEM = "CONICUR", POINTS = number of points,
RADIUS = radius, ANGLES = first point, last point,
SPACE = first end spacing, last end spacing,
COREOUT = segment number $.

Here POINTS and SPACE can be set directly or indirectly as discussed above. The spacings here must be relative, so setting spacings from an existing curve, as for "CURDIST", "LINE", and "SCURVE" is not possible with "CONICUR". Such a setting can be accomplished, however, by following the "CONICUR" operation with the "CURDIST" operation discussed above.

The circular arc extends from the first angle to the last angle, each of which is given in degrees measured counter-clockwise from the positive x-axis.

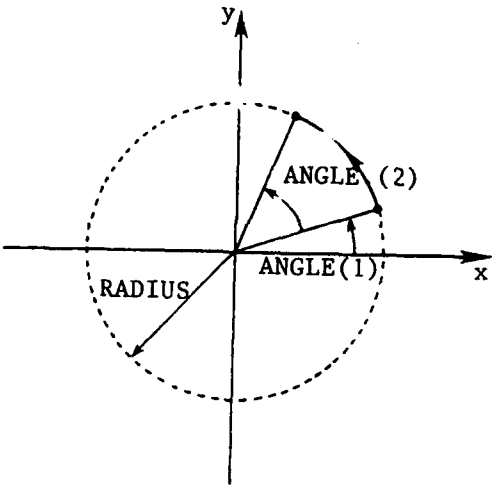


Figure 13. Angle and Direction Definitions

RADIUS and the two entries of ANGLES are real numbers. Assigning a segment number by including COREOUT is, as always, optional. There are other possibilities with the operation "CONICUR", including elliptical, parabolic, and hyperbolic arcs, for which Volume II should be consulted. ted.

Composite Curves

It is possible to concatenate curves by the sequence of operation statements

```
$INPUT ITEM = "CURRENT", COREIN = first segment $
$INPUT ITEM = "INSERT", COREIN = next segment $
.
.
.
$INPUT ITEM = "INSERT", COREIN = last segment,
    COREOUT = segment number $
```

Here the first end of the curve segment indicated on the first "INSERT" statement is attached to the last end of the curve segment indicated on the "CURRENT" statement, the attachment points being overwritten. Each succeeding curve segment is attached to the latest combination in like manner:

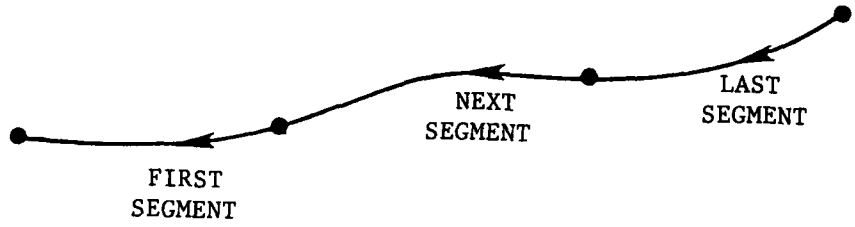


Figure 14. Space Curve Using INSERT

A segment number can be assigned to the combination by the (optional) inclusion of COREOUT. The concatenation is simply additions to an array, and the code does not establish, or check for, any continuity not inherent in the combination. There are other possibilities with operation "INSERT", and Volume II should be consulted.

Insertion

A segment can be inserted into another segment by the operation statements

```
$ITEM = "CURRENT",  COREIN = receiving segment $
$ITEM = "INSERT",   COREIN = inserted segment,
START = starting point,  COREOUT = resulting segment $
```

Here the segment indicated by COREIN on the "INSERT" statement is inserted (overwritten) into the segment indicated on the "CURRENT" statement, beginning at the point where indices are specified by START (Indirect addressing can be used for START). The result may be given a segment number by COREOUT as usual. This operation can be used for curves or surfaces, and can also be used to concatenate surfaces.

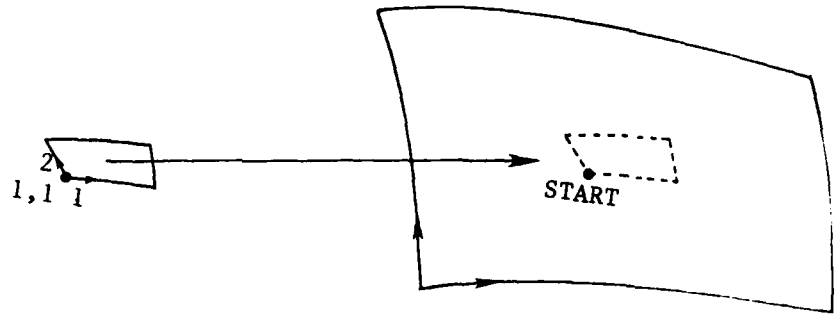


Figure 15. Insertion of a Segment

Extraction

A segment can be extracted from an existing segment by the operation statement

$INPUT ITEM = "EXTRACT", COREIN = existing segment,

START = starting point, POINTS = dimensions of extraction,

COREOUT = segment number $.

The existing segment is obtained via COREIN, and a section whose size is defined by the two entries of POINTS is extracted starting at the point defined by the two entries of START:

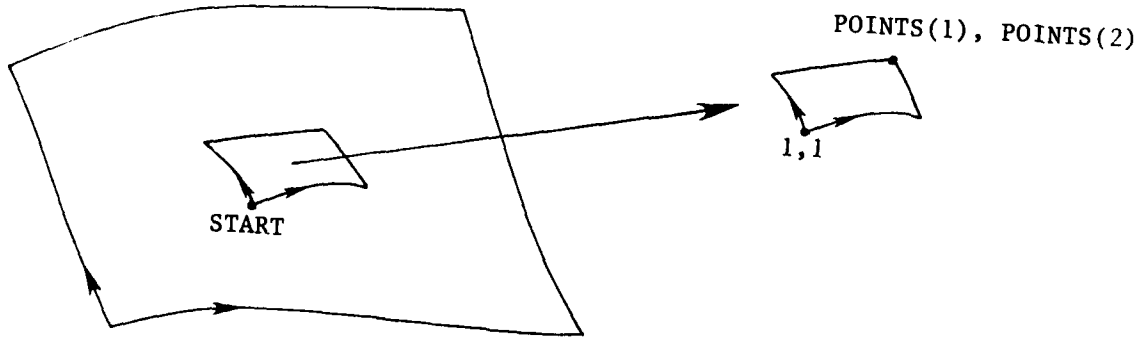


Figure 16. Extraction of a Portion of a Surface

Indirect addressing can be used for both POINTS and START. Again the assignment of a segment number to the extracted section via the inclusion of COREOUT is optional.

Transformations

Curve and surface segments can be transformed by the operation statement

E$INPUT ITEM = "TRANS", COREIN = original segment number,

COSINES = transformation matrix,

COREOUT = transformed segment number $

The transformation matrix, given as nine entries of COSINES, is composed of the direction cosines of the old axes in the new system. The first three entries are the dot products of the old $\bar{i}$ vector with the new $\underset{\sim}{i},\underset{\sim}{j},\underset{\sim}{k}$ vectors; etc.:

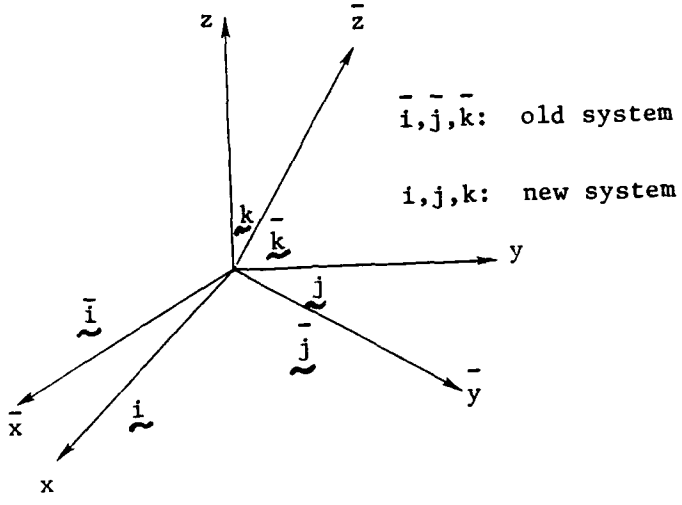


Figure 17. Coordinate Unit Vectors

$$\text{COSINES} = \begin{bmatrix} 1 & 2 & 3 \\ \bar{i} \cdot \underset{\sim}{i} & \bar{i} \cdot \underset{\sim}{j} & \bar{i} \cdot \underset{\sim}{k} \\ 4 & 5 & 6 \\ \bar{j} \cdot \underset{\sim}{i} & \bar{j} \cdot \underset{\sim}{j} & \bar{j} \cdot \underset{\sim}{k} \\ 7 & 8 & 9 \\ \bar{k} \cdot \underset{\sim}{i} & \bar{k} \cdot \underset{\sim}{j} & \bar{k} \cdot \underset{\sim}{k} \end{bmatrix}$$

As always, actual angles (in degrees) can be given in place of cosines if desired.

If translation as well as rotation is involved, the origin of the old system should be placed at a location x_0, y_0, z_0 in the new system before rotation by including

$$\text{ORIGIN} = x_0, y_0, z_0.$$

The segment number of the transformed segment, assigned by COREOUT, can be the same as, or different from, that of the original segment. Any number of segments can be treated by one such transformation operation by giving more than one segment number for COREIN and COREOUT. If COREOUT="SAME", the transformed segments will replace the original segments. In this operation on multiple segments, a negative entry in COREIN or COREOUT implies all segment numbers from the preceding number to the magnitude of the negative number. Thus the following are equivalent:

$$\text{COREIN} = 1,2,3,4 \quad \text{and} \quad \text{COREIN} = 1, -4$$

Scaling

One or more segments can be scaled by the operation statement

$INPUT ITEM = "SCALE", COREIN = original segment number,

SCALE = three scale factors,

COREOUT = scaled segment number $

Here the three scale factors, one for each coordinate, given for SCALE range on -1 to +1. Negative values give mirror-image reflections. Several segments can be treated at once, as with the operation "TRANS". (SCALE can be included on "TRANS", as well, with the scaling being done first.)

Surface by Rotating Curves

An axis system containing a curve segment can be rotated about an axis through the origin of the system to sweep out a surface, e.g. a body of revolution, by the following sequence of the operation statements:

```
$INPUT ITEM = "BOUNCUR", COREIN = segment number $
$INPUT ITEM = "BOUNCUR", COREIN = segment number $ (optional)
$INPUT ITEM = "ROTATE", ANGPTS = number of angles,
     ANGLES = first angle, last angle,
      AXCOS = three direction cosines of rotation axis,
    COREOUT = segment number $
```

If the second "BOUNCUR" statement is omitted, the curve segment indicated by the first "BOUNCUR" statement is rotated to each of ANGPTS angular positions from the first entry of ANGLES to the second.

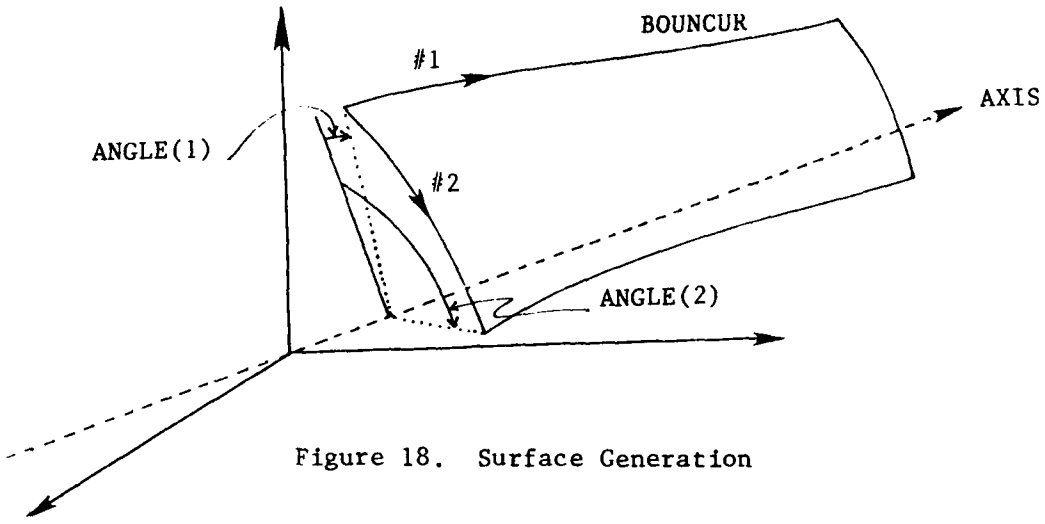


Figure 18. Surface Generation

Note that the first (faster running) direction is along the curve, while the second is in the rotation direction. Here ANGPTS can be set by indirect addressing in the manner of POINTS. These angular positions will be distributed linearly between the first and last angles with the operation statement as given. The relative angular spacings at each end can be set by including SPACE (with direct setting or indirect addressing). Actual angles can be given in place of the direction COSINES if desired. The rotation is clockwise looking down the rotation axis (right-hand rule). With the second operation "BOUNCUR" included, an interpolation is done between the two bounding curves before the rotation. This interpolation follows the same relative distribution used for the angular positions. The two curves must be placed in the same location in space with a "TRANS" operation before being designated as bounding curves.

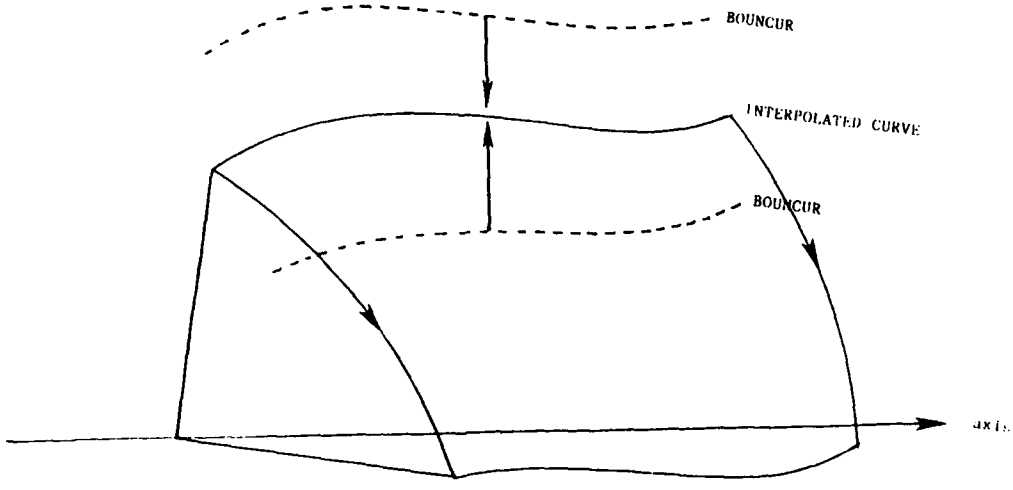


Figure 19. Rotation Using Bounding Curves

It is possible to create a body of revolution with an angular variation of radius by first creating and storing a function of radius in terms of angle, $r(\theta)$. On the "ROTATE" operation, DISTYP="CURVE" is included in this case instead of SPACE, and the radial distribution curve is supplied via COREIN. The two bounding curves in this case have the same shape but different sizes. Volume II can be consulted for other possibilities.

Surface by Stacking Curves

Curve segments can be stacked along a space-curve axis to form a surface segment, e.g. a curved duct, by the following series of operation statements

```
$INPUT ITEM = "BOUNCUR", COREIN = segment number $

$INPUT ITEM = "BOUNCUR", COREIN = segment number $ (optional)

$INPUT ITEM = "AXIS", COREIN = segment number $

$INPUT ITEM = "STACK", AXPTS = number of axial positions,
      AXCOS = direction cosines of axis tangent in curve system,
    NORCOS = direction cosines of axis principal normal in curve
             system,
   COREOUT = segment number $
```

Here one or two bounding curve segments are designated as for the operation "ROTATE". The operation "AXIS" designates a curve segment as the axis of the stack. If only a single bounding curve is designated, then the axes system containing this curve is placed with its origin at a point on the axis of the stack, with the axis tangent at that point aligned with the AXCOS vector in the curve axes system, and with the principal normal to the axis aligned with the NORCOS vector.

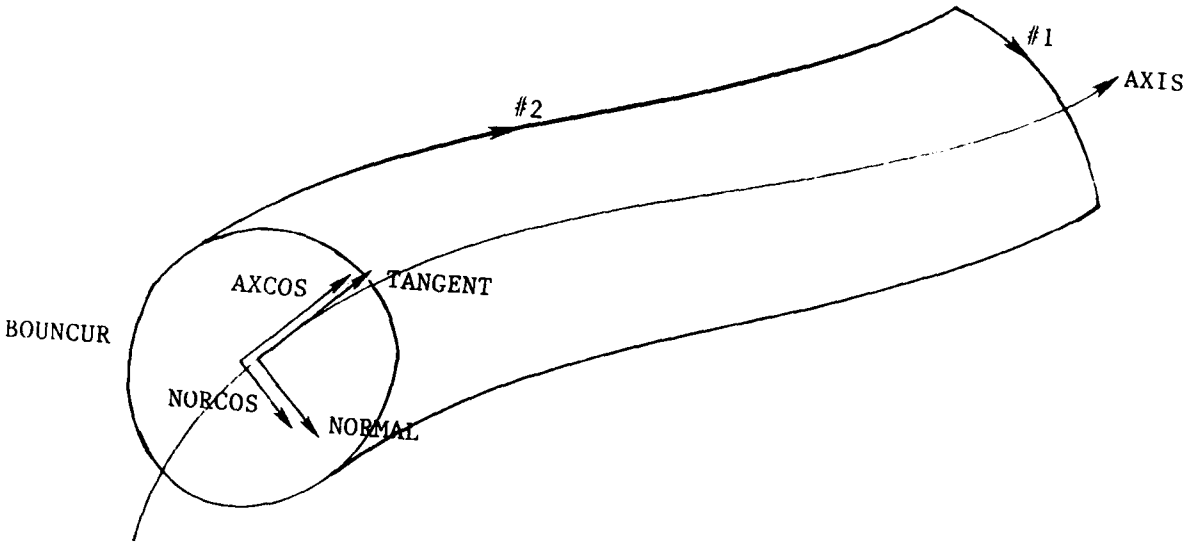


Figure 20. Surface Generation by Stacking

The positions on the axis are simply the points on the axis curve. Note that the first (faster running) direction is along the curves, while the second is along the axis. If two bounding curves are given, both must be on the same axes system. Interpolation is done between these two curves, and then the interpolated curve is placed in position on the axis. The interpolation will be linear with axial arc length unless SPACE is included to set a relative distribution. Again indirect addressing can be used for SPACE.

With two bounding curves it is possible to construct such things as a duct or body that transitions, for instance, from a circular cross-section to an elliptical cross-section:

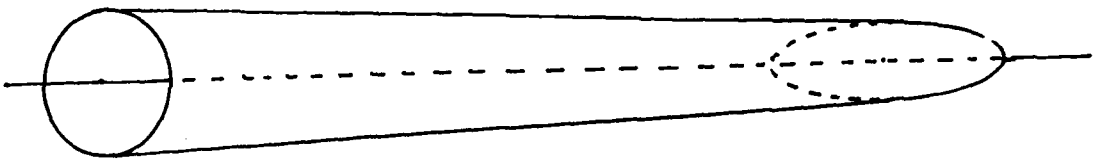


Figure 21. Surface Generation Using Circle and Ellipse

It is also possible to create a duct or body with a constant cross-sectional shape but having a specified variation of radius with axial arc length:

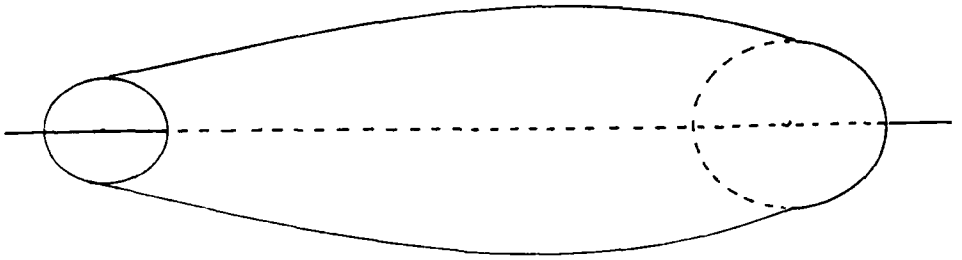


Figure 22. Example of a Stack to Generate a Surface

This is accomplished by creating and storing a curve defining the radius as a function of axial arc length. On the "STACK" operation, DISTYP = "CURVE" is included instead of SPACE, and the radial distribution curve is supplied via COREIN. The two bounding curves in this case have the same shape but different sizes. Volume II can be consulted for other possibilities.

Surface by Blending Curves

A surface segment can be created by interpolation between two curve segments by the following series of operation statements:

```
$INPUT ITEM = "BOUNCUR", COREIN = segment number $

$INPUT ITEM = "BOUNCUR", COREIN = segment number $

$INPUT ITEM = "BLEND", CURVES = number of curves, SPACCUR=relative
              spacing,

    COREOUT = segment number $
```

Here both bounding curves are required. The number of interpolated curves is given by CURVES (with indirect addressing possible as with POINTS), including the two bounding curves.

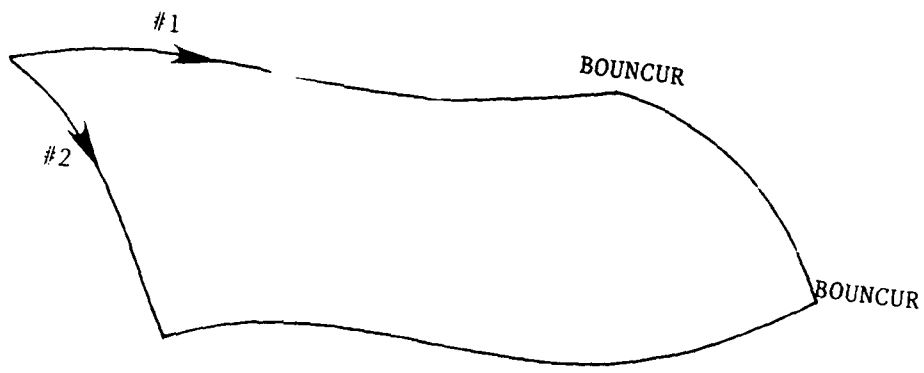


Figure 23. Surface Generation Using BLEND

Note that the first (faster running) direction is along the curves, while the second is in the interpolation direction. The interpolation is linear unless SPACCUR is included to set a relative distribution. Again indirect addressing can be used for SPACE. (There are other possibilities, and Volume II should be consulted.)

Surface by Transfinite Interpolation

A surface segment bounded by four curve segments can be generated by transfinite interpolation via the following set of operation statements:

```
$INPUT ITEM = "EDGECUR", EDGE = "LOWER1",
     COREIN = curve segment number $

$INPUT ITEM = "EDGECUR", EDGE = "UPPER1",
     COREIN = curve segment number $

$INPUT ITEM = "EDGECUR", EDGE = "LOWER2",
     COREIN = curve segment number $

$INPUT ITEM = "EDGECUR", EDGE = "UPPER2",
     COREIN = curve segment number $

$INPUT ITEM = "TRANSUR", COREOUT = surface segment number $
```

here the four curve segments forming the edges of the surface must have already been generated and stored, and opposite edges must have the same number of points. These four curves must meet at four corners, of course. The edges are as follows:

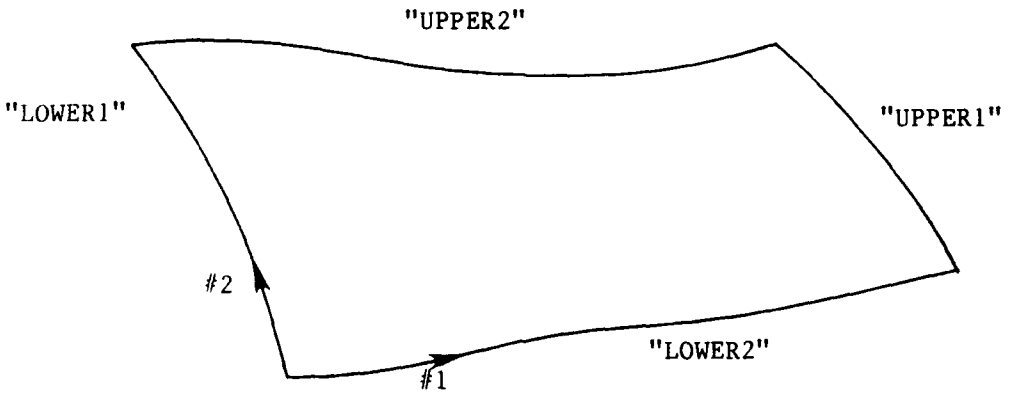


Figure 24. Surface Generation by Transfinite Interpolation

where, as usual, the #1 direction is the faster running direction.

Intersection of Two Surfaces

The intersection curve between two surfaces can be generated by the following operation statements after the two surfaces have been generated:

```
$INPUT ITEM = "CURRENT", COREIN = male surface segment $
$INPUT ITEM = "INTSEC", COREIN = female surface segment,
    COREOUT = intersection curve $
```

The intersection curve is composed of the intersection points of the #2 direction lines on the male surface with the female surface:

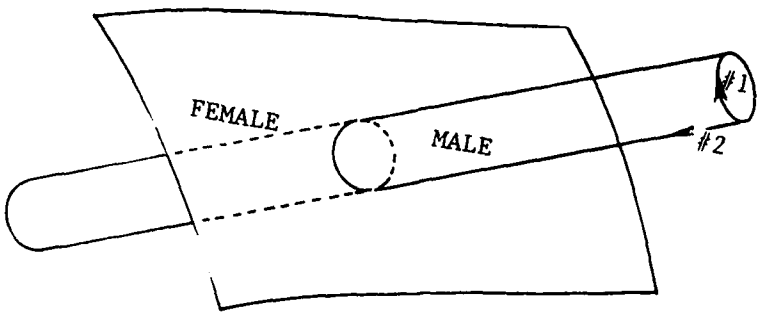


Figure 25. INTSEC Operation

If the male surface was generated with the directions opposite from that needed here, the #1 direction can be made the #2 direction by the operation statement

$INPUT ITEM = "SWITCH" $

inserted immediately before the "INTSEC" statement.

Switching Directions

The "SWITCH" operation can be used in general, not only to switch the roles of the directions, but also to reverse the point progression in either or both directions by the inclusion of REORDER with up to three entries. The three possible entries are "SWITCH" which switches the two directions (the default), and "REVERSE1" and "REVERSE2" which reverse the point progression in the two directions. The order of the entries is irrelevant, and switching is always done first if called for. A segment can be supplied via COREIN, and can be stored with a different segment number via COREOUT if desired.

c. Construction on Curved Surfaces

Some of the operations in the boundary code can be applied on curved surfaces in terms of surface parametric coordinates, e.g. longitude and latitude on the surface of the Earth. In this mode the operation

"LINE", for instance, will construct a curved line lying on the surface between two points on the surface. This mode is invoked by including SURFACE="CURVED" on the "LINE" operation statement. The surface must, of course, have been generated before the "LINE" operation is used, and also the operation statement

```
$INPUT ITEM = "SPLINE", COREIN = surface segment $
```

must be given before the "LINE" operation in order to spline the surface. The cubic curve operation "SCURVE" can also operate in this mode.

A grid lying on a surface can be created by interpolation between two curves on the surface via the "BLEND" operation, or by transfinite interpolation from four edges via the "TRANSUR" operation. In each case, the bounding curves must have been generated to lie on the surface, e.g. by "SCURVE" or "LINE", as discussed above, or as the intersection of the surface with another surface. The "BLEND" or "TRANSUR" operation statements must include SURFACE="CURVED", and the surface must have been splined via the "SPLINE" operation statement discussed above before prior to all of these operations.

For example, such a grid can be constructed on the male surface after an intersection using the intersection curve as a grid line as follows. With the two intersecting surfaces created as usual, and the intersection curve generated by the operation statements

```
$INPUT ITEM = "CURRENT", COREIN = male surface $
$INPUT ITEM = "INTSEC", COREIN = female surface,
    COREOUT = intersection curve $
```

the male surface is splined by

```
$INPUT ITEM = "SPLINE", COREIN = male surface $
```

and then the intersection curve is made by a bounding curve for the "BLEND" operation by

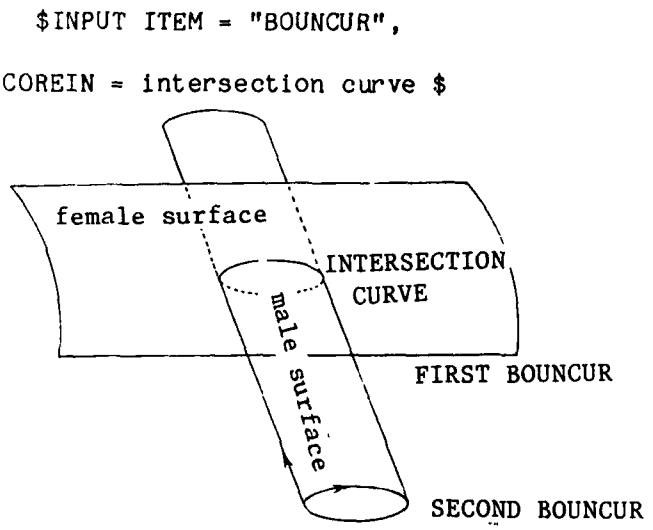


Figure 26. Intersection Curve to Generate a Surface

Next the curve forming the edge of the male surface opposite the intersection is extracted by the operation

$INPUT ITEM = "EXTRACT", COREIN = male surface,

START = first point on edge, POINTS = number of points on edge $

Here two entries are given for START, i.e., the indices of the first point on the edge opposite the intersection. This extracted curve is then made for the second bounding curve by

$INPUT ITEM = "BOUNCUR" $

(No COREIN is necessary here since the curve intended is the result of the immediately preceding operation.) The operation

$INPUT ITEM = "BLEND", SURFACE = "CURVED",

CURVES = number of curves, COREOUT = grid segment $

then generates the desired grid on the male surface between the intersection and edge curve.

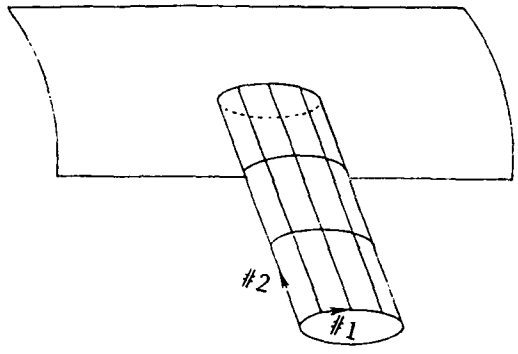


Figure 27. Example of Use of Intersection Curve

A grid on this male surface could also have been generated by transfinite interpolation on the surface via the "TRANSUR" operation, with the four edge curves being the intersection and three edge curves extracted from the male surface.

This capability for the generation of a grid on a curved surface allows the boundary code to function as an algebraic surface grid generation system, but the primary purpose of surface grids here is for input as boundary segments to the grid code.

d. Output

After all of the surface segments to be input to the grid code have been generated, a file of these segments can be created by the operation

```
$INPUT ITEM = "COMBINE", CONTENT = "YES",
    COREIN = surface segments, FILEOUT = file number $
```

Here the surface segment numbers are given for COREIN, in any order and separated by commas. Sequences of consecutive numbers can be given as the first number in the sequence followed by a comma and then by the negative of the last number in the sequence. The file indicated then can be preserved for input to the grid code.

A similar usage of the "COMBINE" operation, but with HEAD="YES", TRIAD="YES", FORM="E" included instead of CONTENT, will create a formatted file of the segments indicated that can be read by a plot code. The form of this file is

```
   1    M    N
   x    y    z
   x    y    z
        .
        .
        .
---------------
   2    M    N
   x    y    z
   x    y    z
        .
        .
        .
---------------
      etc.
```

where the numerals appear as given, and MxN is the segment size. The coordinates appear in 3E20.8 format.

e. Other Operations

There are a number of other operations, and variations of the operations discussed here, in the boundary code, and Volume II should be consulted. The present discussion is not meant to be a full reference, but is intended as a general introduction to the use of the code, particularly in concert with the grid code.

3. GRID CODE

The grid code receives the Cartesian coordinates of points on boundary segments and generates the grid within the physical region defined by these boundaries. These boundary points can have been generated by the boundary code (or another code) to be read from a file by the grid code, or can be input directly in the grid code runstream.

The grid is constructed of six-sided (four in 2D) contiguous blocks which fit together to fill the entire physical region. The number, size, and arrangement of these blocks is arbitrary and is set up in the grid code runstream. In general, the grid is generated to be completely continuous across the block interfaces. These blocks have curved sides in the physical region, of course, but correspond to rectangular blocks in the computational region. An extra layer of points surrounding each block is used by the code to establish an image-object correspondence across the block interfaces to provide this continuity. Each block may be of a different size, and the only requirement is that they fit together to fill the physical region.

Each block has its own set of curvilinear coordinates (ξ^1,ξ^2,ξ^3), which assume integral values at the grid points, varying from unity to the number of points in each direction in the block:

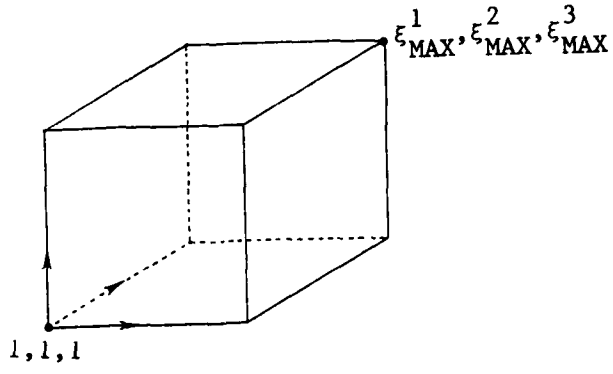


Figure 28. Curvilinear Coordinates

(These curvilinear coordinates are thus synonymous with the I,J,K indices of DO loops over the block.) Although the grid lines are continuous across the block interfaces in the physical field, the species (and/or direction) of the curvilinear coordinates can change across the interface:

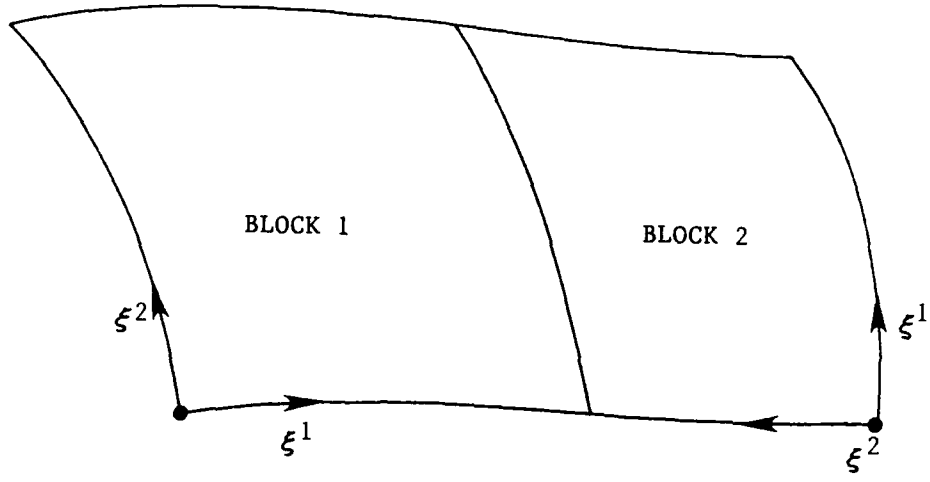


Figure 29. Curvilinear Coordinates Across Block Boundaries

Although it is not required, the boundary segment numbers assigned in the boundary code can be transferred to the grid code via the file that contains the Cartesian coordinates of the boundary points. This, and the (optional) use of the point numbers assigned in the boundary code runstream, allows changes in the number of points on the various segments to be accomplished through very localized changes in the runstreams (with no changes required in the grid code runstream in most cases).

The first step in this use of numbered points and segments (indirect addressing) is the making of good sketches of the boundary segments and the block structure. All points defining the edges of the boundary segments and the corners of the blocks, together with other important points, should be numbered on these sketches. Similarly, the boundary segments that are to be transferred from the boundary code to the grid code should be numbered. As noted in the discussion of the boundary code, it is helpful to adopt some grouping of these numbers to identify certain naturally related points and segments. Since many points and segments will be shared by different blocks, point and segment numbers

should be unique to physical position, with the same numbers being used for shared points and segments in different blocks. Finally, the directions of the curvilinear coordinates in each block should be indicated on the sketches.

The grid code input runstream is similar in form to that of the boundary code, directing the code through a sequence of operations through successive NAMELIST input statements of the form

$INPUT ITEM = "__", --- $

with the particular operation being identified by the name within the quotes, and the quantities relevant to that operation included on the statement.

a. Reading Boundary Segments

There are two basic modes for transferring the boundary segments generated in the boundary code to the grid code. These segments can either be read from a file one at a time, or can all be read from the file at once. The former mode does not involve coupling of the boundary and grid codes through the use of segment numbers, and in that case CONTENT="YES" should not be included on the ITEM="COMBINE" operation statement in the boundary code runstream since no segment numbers are involved. This is also the mode that is used when the boundary segments are generated by some means other than the boundary code. The other mode, reading all the segments at one time from the file, is used when the boundary and grid codes are coupled through the use of segment numbers, with the file having been generated by the "COMBINE" statement in the boundary code with CONTENT="YES" included. The segment numbers are the numbers assigned by COREOUT in the boundary code.

Boundary surface segments are read in as rectangular arrays of points, with three Cartesian coordinates per point. The segment is read onto a section defined by two points in the block, which are identified by START and END on the operation statement that does the reading. This

section in the block has the points identified by START and END at opposing corners and must be of the same size as the rectangular array being read in.

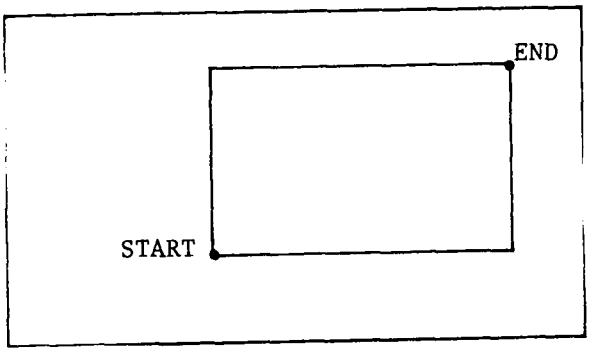


Figure 30. Segment Read

START and END each can be given either as the three indices of tne point in the block, or as the point number if the segment number coupling (indirect addressing) is used. One index for START must, of course, be the same as the corresponding index of END in the former case since the segment forms a surface. (Two corresponding indices will be the same when a curve is read.) The reading is done from START to END, and the direction (1,2, or 3) in the block that is to run fastest (i.e., the inner loop) in the reading is identified by ORDER. It is not necessary to include ORDER if the fastest and slowest directions are in the order 1,2,3, or if the segment number coupling is used, or if the segment is a curve instead of a surface.

No Coupling with Boundary Code

Reading one boundary segment at a time is done by the operation statement

```
$INPUT ITEM = "FILE", FILE = file number,
START = first point, END = last point $
```

Here START and END each are three integers (two in 2D) supplying the indices, i.e., the curvilinear coordinate values, of the two opposite corners of the segment being read, as has been discussed above. One such operation statement is given for each boundary segment read in this manner. If more than one of these segments is on a single file, it is obviously necessary that the segments be read in the order in which they appear on the file. The file here simply contains the three Cartesian coordinates (two in 2D) for each point on a boundary segment. Several formats are available, and Volume III should be consulted. The default is the three coordinates of a point together as a triad on a line, one line for each point, unformatted. (A similar operation statement, but with "FILE" replaced by "LIST" and FILE replaced by VALUES, reads segments directly from the NAMELIST. Here the three (two in 2D) Cartesian coordinates of each point on the segment are given in succession as the entries of VALUES, these real numbers being separated by commas. Spaces can be included and any pattern is allowed. Again, one such operation statement is given for each segment read in this manner.)

Coupling with Boundary Code

When the boundary and grid codes are coupled through the transfer of segment numbers on the file as well, all of the boundary segments on the file are read at one time. (Here the file must have been generated by the boundary code via the ITEM="COMBINE" operation statement with CONTENT="YES" included, as discussed above.) In general this reading is done by the operation statement

```
$INPUT ITEM = "FILE", FILE = file number,
       ALL = "YES" $
```

which should be preceded in the input runstream only by the ITEM = "INITIAL" operation statement (discussed later).

b. Block Definition

Each block is introduced in succession in the runstream by the statement

$INPUT ITEM = "BLOCK" $

The code numbers the blocks as they are introduced. If the segment number coupling with the boundary code is not used, this statement should include

SIZE = block dimensions

giving the number of points in the three (two in 2D) directions in the block as three (two in 2D) integers.

Point Locations - Coupling with Boundary Code

If segment number coupling is used, the location of each numbered point in the block is set next by a series of operation statements with ITEM="POINT". The first of these must be

$INPUT ITEM = "POINT", POINT = point number, LOCAT = point location $

The point location here is three (two in 2D) integers defining the indices (curvilinear coordinates) of the point in the block. This point is typically the one at (1,1,1), but can be anywhere in the block. The other numbered points are then set in relation to previously set points by statements of the form

$INPUT ITEM = "POINT", POINT = point number,
OPOINT = other point number, SEGMENT = segment number,
DIRECT = direction on grid, NDEX = direction on segment $

Here the point being set and the "other point" are both end points of an edge of the segment named. DIRECT is ±1,±2, or ±3, indicating the curvilinear direction in the block from this other point to the point being set, and NDEX is 1 or 2 indicating whether the segment edge was the faster (#1) or slower (#2) direction on the segment when the segment was generated by the surface code.

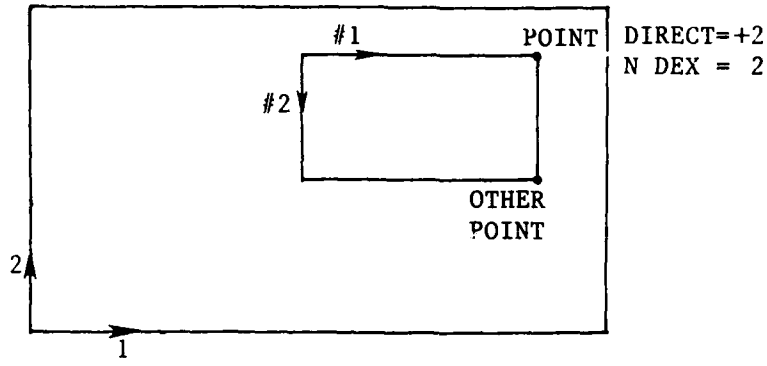


Figure 31. Point Locations

If the segment is a curve, instead of a surface, NDEX is omitted. All the numbered points can be set by a succession of such statements in a progression through the block. This progression is not unique, of course, but the indices assigned are unique.

It is possible to set the indices of a group of points from another corresponding group, e.g. on different meridional planes of a body of revolution. The operation statement

```
$INPUT ITEM = "POINT", POINT = increment, SEGMENT = segment number,
    DIRECT = direction in block, NDEX = direction on segment $
```

sets a new group of points from all of the currently set points in the block. The numbers of the points in the new group are formed by adding the value given for POINT to the current point numbers.

Once these indices have been assigned in this manner, the points can be referenced by giving the point number as a single positive integer, in place of the three indices of the point, for START,END,ISTART or IEND

on all statements requiring the point. Values of individual indices from a numbered point can be used to reference other unnumbered points by giving negative integers for one or more entries of START,END,ISTART or IEND. In this case the value of the index will be that of the same index for the numbered point indicated by the magnitude of the negative value. Thus, the circled point in the illustration below

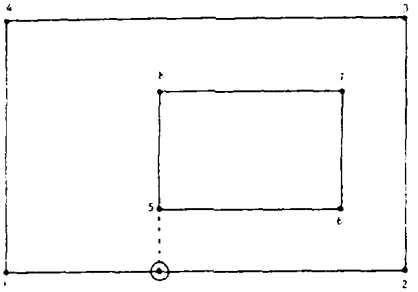


Figure 32. Indirect Point Referencing

can be referenced by START=-5,-1 (or equivalently by -8,-2). Positive and negative entries for these quantities may be mixed, with the former giving actual values of the indices.

The point assignment statements will appear for each block, following the ITEM="BLOCK" statement for each, so points common to more than one block will have the same number in each block, but will be assigned appropriate different indices in each block.

Placing of Segments - Coupling with Boundary Code

Next in this segment number coupling mode comes a series of statements that locate the boundary segments in the block:

$INPUT ITEM = "SEGMENT", SEGMENT = segment number,

START = first point number, END = last point number $

Here START and END define the first and last points on the segment as it was generated by the surface code:

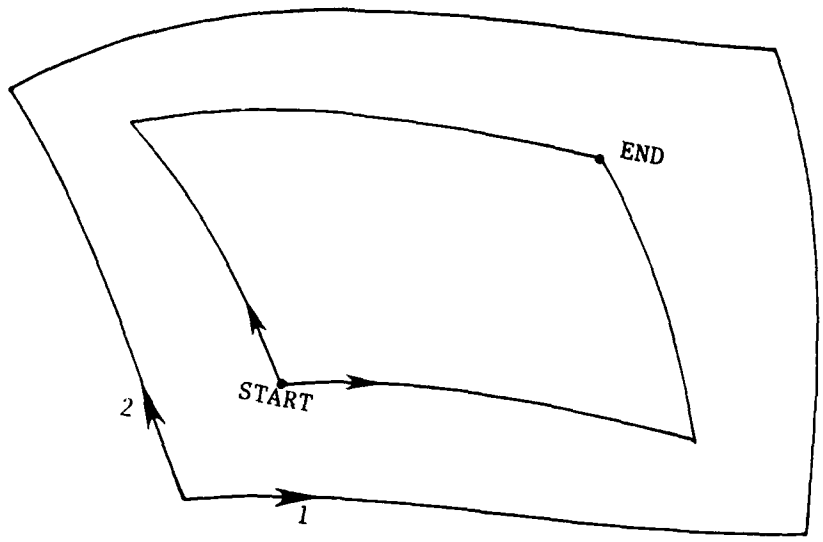


Figure 33. Segment Specification

It is not necessary to give the "SEGMENT" statements in the order of the segments on the file, and the same segment can be placed at different positions in the block, e.g., for the two sides of a cut.

Block Size - Coupling with Boundary Code

Finally the block size is set by the statement

$INPUT ITEM = "SIZE", SIZE = point number $

where the point number given is that of the point opposite the point (1,1,1).

No Coupling

If the segment number coupling mode is not used, the block size is set on the "BLOCK" statement as has been discussed above, and the "SEGMENT" statements are replaced by the "FILE" and/or "LIST" statements also discussed above. It is possible to combine the two modes, placing some segments by "SEGMENT" statements and reading others individually from files via "FILE" statements. The only requirement is that all of the segments placed by "SEGMENT" statements, or used on "POINT" statements, be included on the file generated with CONTENT="YES" by the sur-

face code and read with ALL="YES" by the grid code, as described above. Points can be referenced throughout the runstream either by the single point number or by the three (two in 2D) indices of the point.

Boundary Segments and Interfaces

Boundary segments must be read in for all parts of the physical boundaries of the region, of course. In addition, all edges of each block must be read in if not included as part of another segment. The interfaces between blocks can also be read in, if desired, but this is not necessary since the grid code can interpolate (transfinite) from the edges for points on the interfaces. It does save time in repeated runs of the grid code, however, to read in the interfaces. It may also be desirable to read in the interfaces for purposes of controlling the grid line spacing and orientation.

c. Boundary Point Types

Fixed Points

All points that are to remain fixed as the grid is generated must be so designated either as a segment is placed (via the "SEGMENT", "FILE", or "LIST" operations), by including CLASS="FIX" on the operation statement, or must be explicitly designated by separate operation statements of the form

```
$INPUT ITEM = "FIX", START = first point, END = last point $
```

with START and END identifying a section of points in the block as usual. Once a point has been designated "FIX", that designation can be changed only by an operation statement similar to that above but with "UNFIX" in place of "FIX". On these statements, and on all others to be discussed, START and END can each be given as the three (two in 2D) indices of the point or as the point number.

Boundary Orthogonality

The grid can be made orthogonal on sections of surfaces (curves in 2D) in a block by the operation statement

```
$INPUT ITEM = "ORTHOG",
START = first point, END = last point $
```

This will cause the grid to be orthogonal on the section identified by START and END without moving the points on the section. If this section is in the interior of the block, the orthogonality will occur only on the upper side. Orthogonality on the lower side is set by a similar statement with DIRECT="LOWER" included.

Symmetry Boundaries

Symmetry conditions can be imposed on a section of a plane boundary by an operation statement of the above form, but with "REFLECT" in place of "ORTHOG".

Sub-Blocks

The grid code generates an algebraic grid by transfinite interpolation from all the faces of a block. Internal faces within the block can be imposed, if desired, for this interpolation by creating sub-blocks within which the interpolation is done separately. Such a sub-block structure is created by an operation of the form

```
$INPUT ITEM = "SUB",
START = first point, END = last point $
```

where START and END identify points at two opposing corners of the sub-block. It is necessary that enough of these statements be given to completely fill the block with sub-blocks. Points on the internal faces can either be read in, if desired, or will be interpolated by the code.

The sub-block structure is involved only in the generation of the algebraic grid and in the evaluation of the control functions for the elliptic grid generation system, and does not affect the block structure for storage. The sub-block interfaces do not have to be designated as cuts.

Interfaces

After each block has been set up, interfaces between blocks must be designated as cuts by operation statements of the form

```
$INPUT ITEM = "CUT", BLOCK = object block,
START = first object point, END = last object point,
IBLOCK = image block,
ISTART = first image point, IEND = last image point $
```

Here START and END identify opposing corner points of the cut section on a face of the object block, while ISTART and IEND identify the corners of the corresponding interface on the image block. Each of these can be given as three (two in 2D) indices of the point or as the point number. The point progression from START to END must correspond to that from ISTART to IEND. Also if the index (curvilinear coordinate) species do not correspond between the two blocks at the interface, ORDER must be included. The three (two in 2D) entries of ORDER are a permutation (not necessarily cyclic) of 1,2,3, with the first entry indicating which direction on the object block corresponds to the ξ^1 direction on the image block, etc. The elliptic grid will be continuous across cuts. A cut section does not have to cover an entire block face.

Boundary Points

Points previously designated "FIX" are not affected by "ORTHOG", "REFLECT", or "CUT" designations and can be included in such sections with no effect. Therefore the "FIX" designation must not be attached to points intended for orthogonality or symmetry.

Other Parameters

In addition to the operations described above, there are a few other items to be set. The first operation statement to be given is

```
$INPUT ITEM = "INITIAL", --- $
```

If an algebraic grid only is intended, ITMAX=0 is included on this statement, while for an elliptic grid the maximum number of iterations is given for ITMAX. In the latter case, an iteration convergence tolerance can also be set by including a real number (less than unity) as TOL. If TOL is omitted, the total number of iterations set by ITMAX will be performed. The value given for TOL is a relative tolerance, defined as a fraction of the length scale (cube root of the total volume of the physical region), at which the iteration stops.

Output

Finally, the grid can be stored on file by the statement

```
$INPUT ITEM = "STORE", FILE = file number $
```

There are several other features and options for which Volume III should be consulted.

Sections of the grid can be printed by operation statements of the form

```
$OUTPUT ITEM = "PRINT", BLOCK = block number, START = first point,
        END = last point $
```

where START and END identify the section in the same manner described above for the other statements. Any number of such statements can be given, in an order. Similar statements, but with "PLOT" in place of "PRINT", write the section on file 8 for later input to a plot code. The

form of this file is the same as that described above for plotting from the boundary code, with each section numbered consecutively regardless of the block from which it came.

Example - No Coupling with Boundary Code

If segment number coupling with the boundary code is not used, or if the boundary segments are input from another source or directly in the grid code runstream, the typical form of the grid code runstream is as follows:

```
$INPUT ITEM = "INITIAL", --- $
$INPUT ITEM = "BLOCK", SIZE = __,__,__ $
$INPUT ITEM = "FILE", FILE = __, START = __,__,__,
          .      END = __,__,__, --- $
          .
          .
          .
     $INPUT ITEM = "SUB", START = __,__,__, END = __,__,__ $
          .
          .
          .
     $INPUT ITEM = "FIX", START = __,__,__, END = __,__,__ $
          .
          .
          .
   $INPUT ITEM = "ORTHOG", START = __,__,__, END = __,__,__,
        CONUPI = ___ $
          .
          .
          .
$INPUT ITEM = "BLOCK", SIZE = __,__,__, $
          .
          .
          .
$INPUT ITEM = "CUT", BLOCK = __, START = __,__,__, END = __,__,__,
     IBLOCK = __, ISTART = __,__,__, IEND = __,__,__, $
     .
     .
     .
$INPUT ITEM = "STORE", FILE = __, --- $
$INPUT ITEM = "END" $
```

```
$OUTPUT ITEM = "PRINT", START = __,__,__, END = __,__,__ $
     .
     .
     .
$OUTPUT ITEM = "PLOT", START = __,__,__, END = __,__,__, --- $
     .
     .
     .
$OUTPUT ITEM = "END" $
```

Not all of these statements will appear in each case, and there are still other options discussed in Volume III.

Example - Coupling with Boundary Code

With segment number coupling with the boundary code, the typical grid code runstream is of the form

```
$INPUT ITEM = "INITIAL", --- $
$INPUT ITEM = "FILE", FILE = __, ALL = "YES" $
$INPUT ITEM = "BLOCK" $
     $INPUT ITEM = "POINT", POINT = __, LOCAT = __,__,__ $
     $INPUT ITEM = "POINT", POINT = __, OPOINT = __, SEGMENT = __,
          DIRECT = __, NDEX = __ $
     .
     .
     .
     $INPUT ITEM = "SIZE", SIZE = __ $
     $INPUT ITEM = "SEGMENT", SEGMENT = __, START = __, END = __, --- $
     .
     .
     .
     $INPUT ITEM = "SUB", START = __, END = __ $
     .
     .
     .
     $INPUT ITEM = "FIX", START = __, END = __ $
     .
     .
     .
```

```
   $INPUT ITEM = "ORTHOG", START = __, END = __, CONUPI = __ $
      .
      .
      .
$INPUT ITEM = "BLOCK" $
      .
      .
      .
$INPUT ITEM = "CUT", BLOCK = __, START = __, END = __,
.     IBLOCK = __, ISTART = __, IEND = __ $
.
.
.
$INPUT ITEM = "STORE", FILE = __, --- $

$INPUT ITEM = "END" $

$OUTPUT ITEM = "PRINT", START = __, END = __ $
.
.
.
$OUTPUT ITEM = "PLOT", START = __, END = __ $
.
.
.
$OUTPUT ITEM = "END" $
```

Again not all of these statements will appear in all cases, and there are other options.

d. Grid on Curved Surface

The grid code also has a surface mode by which a 2D grid can be generated on a curved surface. In this mode the surface on which the grid is to be generated is read in by the statement

```
$INPUT ITEM = "SURFACE",  SIZE = surface dimensions,
          FILE = file number $
```

The presence of this statement, which should be first given, activates the surface mode. Two entries are given for SIZE to define the dimensions of the surface. The file can have been generated by the boundary

code, or some other source. The format can be specified as for other reading of files, but the points must be placed one to a line, with the triad x,y,z on a single line (TRIAD="YES" in the boundary code.)

The grid is generated in terms of surface parametric coordinates, and the input is that for normal 2D operation, including the block structure and all the features discussed above. The values read in on the boundary segments (curves here) simply are two parametric coordinates per point, instead of three Cartesian coordinates. These parametric coordinates are analogous to latitude and longitude on the surface of the Earth, but here are the two indices on the surface:

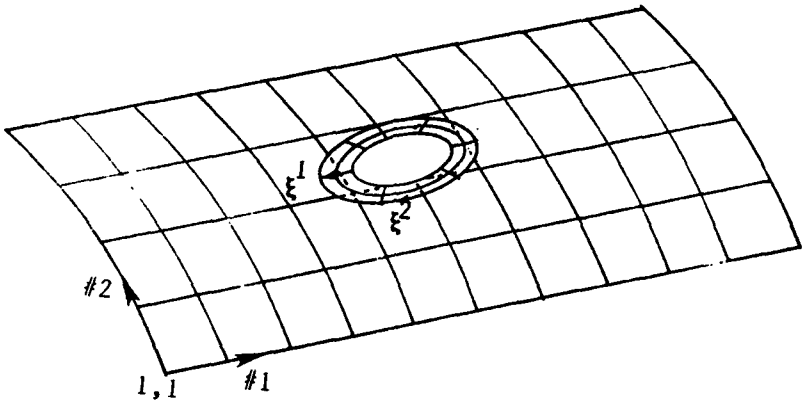


Figure 34. Griding an Existing Surface

The surface is splined in terms of these parametric coordinates, which vary from unity to the dimensions of the surface in each direction. The boundary segments can be put in terms of these parametric coordinates in the boundary code simply by giving the operation statement

$INPUT ITEM = "CORPAR", COREOUT = segment number $

immediately after such a curve segment has been generated on the surface by an operation including SURFACE="CURVED", or as the intersection between the surface and another, in the boundary code.

SECTION V
APPLICATIONS

NACA 0012 AIRFOIL (2-D)

This example will illustrate the procedure for building a two-dimensional C-grid (221X20) about a symmetric airfoil (specifically, an NACA 0012 airfoil). Line-by-line descriptions of the inputs for the boundary code and the inputs for the grid code will be presented as well as simple illustrations of the effect of each line of input.

Figure 35 and 36 are illustrations of the physical space and the computational space. The numbering corresponds to the up-front bookkeeping in this version of the EAGLE code that allows for fast and efficient modifications to the body and grid. Refer to this numbering while going through the following boundary code inputs.

SET POINT CARTESIAN COORDINATES (circled numbers ◯)

```
1.   $INPUT ITEM='POINT', POINT=1, R=30.,0. $
2.   $INPUT ITEM='POINT', POINT=6, R=30.,-30. $
3.   $INPUT ITEM='POINT', POINT=5, R=30.,30. $
```

In Figure 1(a) the x-y coordinate system is as shown. Point #2 will be set by reading in airfoil data to be discussed later. Points #3 and #4 need not be set.

SET NUMBER OF POINTS ON SEGMENTS (squared numbers □)

```
4.   $INPUT ITEM='SETNUM', SEGMENT=9, POINTS=81 $
```

Number of points along SEGMENT #9 which is not shown. To be used later for distribution of points along airfoil.

```
5.   $INPUT ITEM='SETNUM', SEGMENT=1, ITERMS= -9,-9 $
```

Set number of points on the airfoil; total for top and bottom. ITERMS= -9,-9 adds the number of points set for SEGMENT #9 together. That is, 81 + 81 = 161. (Because end point of one is first point of other, the math is actually 81 + 81 = 162 - 1 = 161.)

```
6.   $INPUT ITEM='SETNUM', SEGMENT=2, POINTS=31 $
```

Set number of points on wake line.

```
7.   $INPUT ITEM='SETNUM', SEGMENT=3, POINTS=20 $
8.   $INPUT ITEM='SETNUM', SEGMENT=7, POINTS=141 $
9.   $INPUT ITEM='SETNUM', SEGMENT=6, POINTS=41 $
```

Sets the number of points on the bottom rear outflow surface, front far field surface, and bottom far-field surface, respectively.

```
10.  $INPUT ITEM='SETNUM', SEGMENT=5, ITERMS= -2,-1,-1 $
```

Establishes the number of points for the top half of the computational space to match the sum of the points in the bottom half. 31 + 161 + 31 = 221 [MATH=((ADD-1)-1)].

```
11.  $INPUT ITEM='SETNUM', SEGMENT=8, ITERMS= -5,-6, MATH= 'DIF+1' $
12.  $INPUT ITEM='SETNUM', SEGMENT=8, ITERMS= -8,-7, MATH= 'DIF+1' $
```

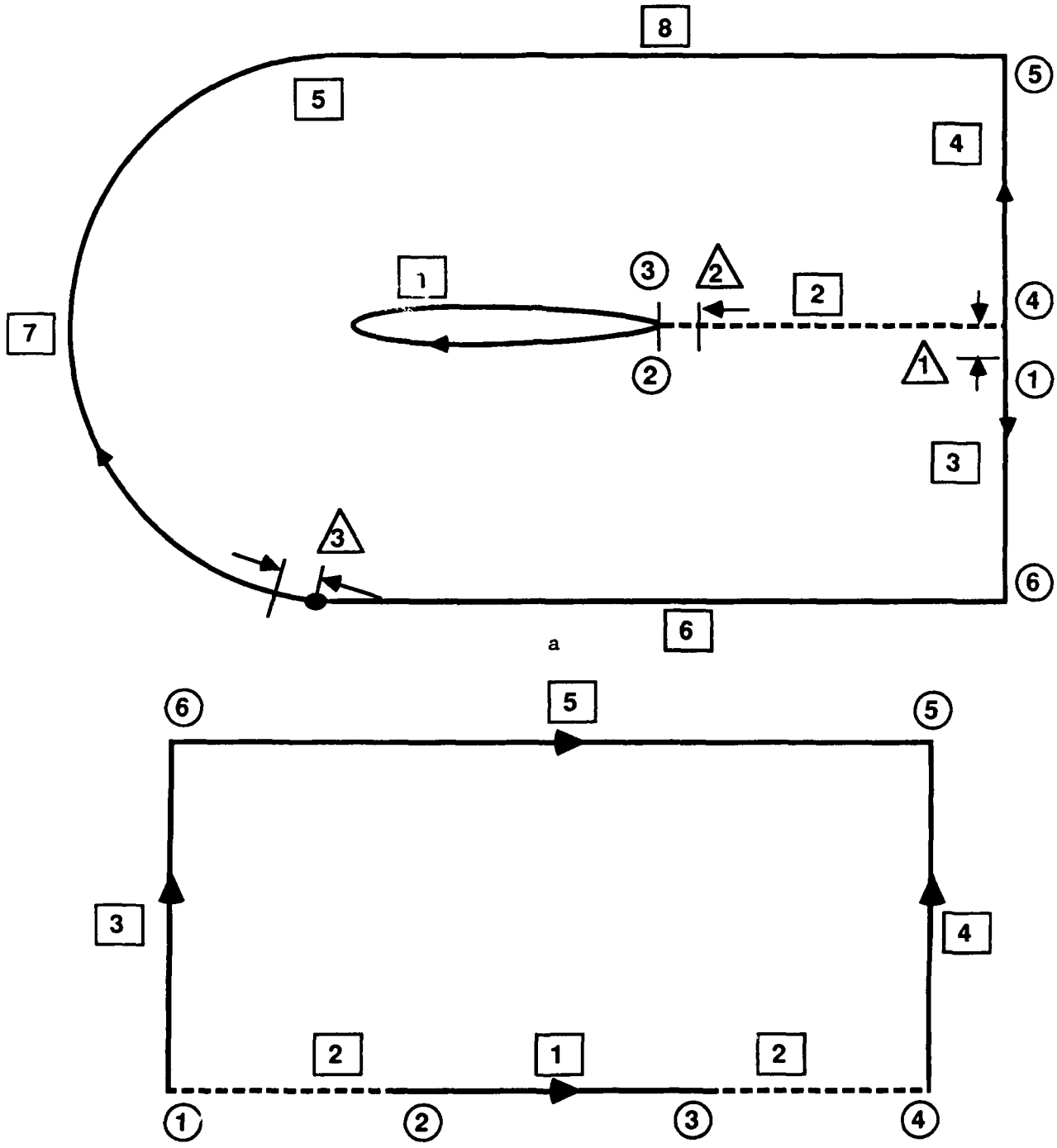


Figure 35. Physical Space (a) and Computational Space (b)

Set the number of points for SEGMENT #8, by first subtracting off SEGMENT #6 and then subtracting off SEGMENT #7. That is, 221 - 41 = 180 + 1 = 181, then 181 - 141 = 40 + 1 = 41. SEGMENT #8 should match SEGMENT #6 in physical space. By setting up the inputs in this fashion, one only need change SEGMENT #6 and SEGMENT #8 will be changed automatically.

SET SPACINGS (triangled numbers △)

```
13.  $INPUT ITEM='SETVAL', NUMBER=1, VALUE=.001 $
14.  $INPUT ITEM='SETVAL', NUMBER=2, VALUE=.01 $
15.  $INPUT ITEM='SETVAL', NUMBER=3, VALUE=.0001 $
16.  $INPUT ITEM='SETVAL', NUMBER=4, VALUE=.0015 $
```

These values can be relative or absolute - see section on distribution parameters.

READ AIRFOIL

```
17.  $INPUT ITEM='CURRENT', POINTS= 500      , VALUES=
18.   0.0000000E+00,  0.0000000E+00,  0.0000000E=00,
                              .
                              .
517.  1.000000      ,  0.0000000E+00,  0.0000000E=00 $
```

Here, x, y, and z values for the top half of the airfoil are read in. Notice the start at the origin and end at x=1.0, y=0.0. All the Z-values are 0.0 for this 2-D case. For this example, all 500 points were not shown. The data was generated by writing a short program using the equation for the line making up the top half of an NACA 0012 airfoil.

DISTRIBUTE POINTS ON THE AIRFOIL

```
518.  $INPUT ITEM='CURDIST', POINTS=-9, SPACE=-4, -2, END= 'LAST', 'LAST',
            COREOUT=9 $
```

Interpolates the 500 points to 81 points using the spacing at the nose, SETVAL 4 and at the trailing edge, SETVAL 2. Save to COREOUT #9.

```
519.  $INPUT ITEM='SWITCH', REORDER='REVERSE1' $
```

'SWITCH' is necessary to get points to run from right to left for building the bottom half of the airfoil.

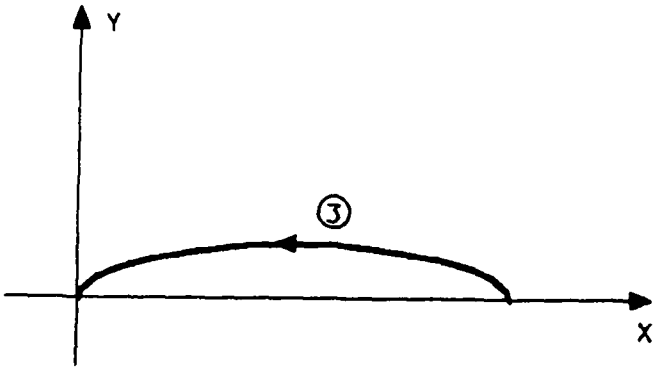


Figure 36. 'SWITCH' for Bottom Half of Airfoil

```
520.  $INPUT ITEM='SCALE', SCALE=1, -1, 1 $
521.  $INPUT ITEM='INSERT', COREIN=9, COREOUT=1 $
```

'SCALE' rotates the line segment by making all the y-values negative. 'INSERT' merges the top and bottom halves of the airfoil and saves it to COREOUT #1.

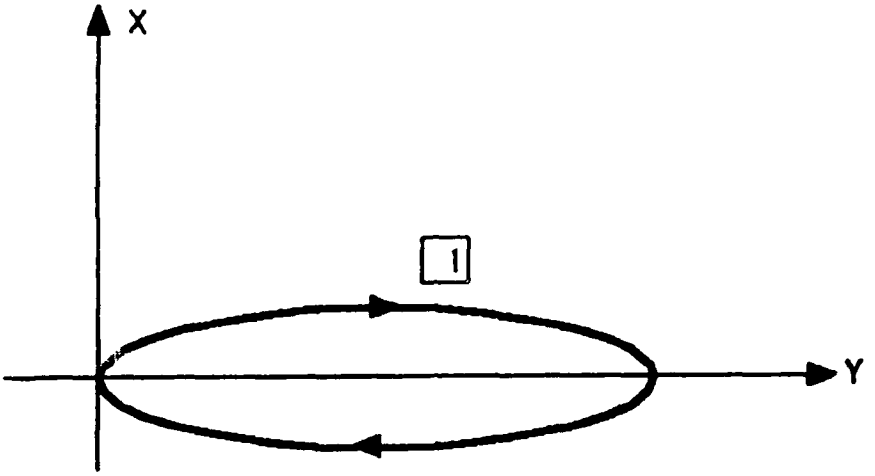


Figure 37. 'INSERT' Merging Top and Bottom Half of Airfoil

BUILD FRONT OUTER BOUNDARY

```
522.  $INPUT ITEM='CONICUR', TYPE='CIRCLE', RADIUS= 30., ANGLE=270,90,
            POINTS=100, COREOUT=7 $
```

Draws a half-circle of radius 30 from 270 degrees to 90 degrees.

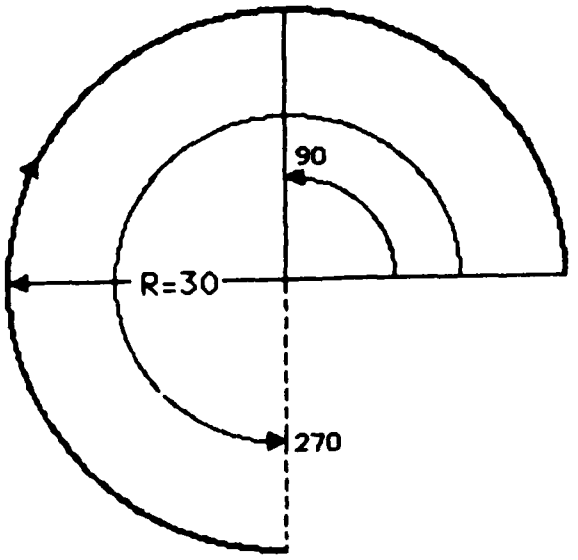


Figure 38. Drawing the Half Circle for the Front Outer Boundary

```
523.  $INPUT ITEM='CURDIST', COREIN=7, POINTS=-7, SPACE=-3, -3, COREOUT=7 $
```

Puts the number of points set for Segment #7 on this curve. Use relative spacing SETVAL 3 on each end. COREOUT number is SEGMENT number.

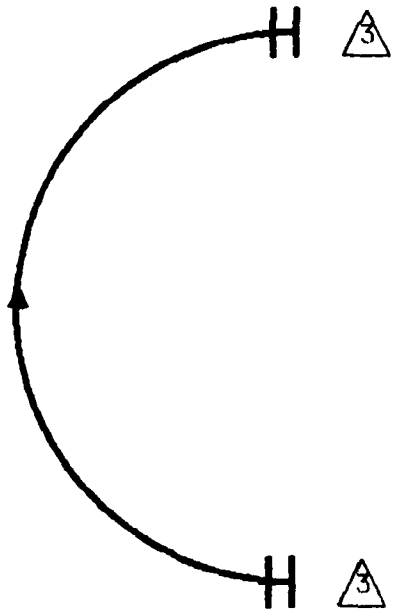


Figure 39. Detail of Spacing for Front Outer Boundary

PLACE POINTS ON SEGMENT #6

```
524.  $INPUT ITEM='GETEND', COREIN=7, POINT='FIRST', END='FIRST' $
525.  $INPUT ITEM='LINE', POINTS=-6, R2=6, SPACE=7, END='FIRST' $
526.  $INPUT ITEM='SWITCH', REORDER='REVERSE1', COREOUT=6 $
```

'GETEND' takes the first point from SEGMENT #7 as R1 for 'LINE'.

'LINE' uses the spacing at the first end (or beginning) of SEGMENT #7 as the spacing at the first end (or beginning) of 'LINE' and uses the same number of points as SEGMENT #6. 'LINE' starts at the first point on SEGMENT #7, retrieved by GETEND, and ends at POINT #6.

'SWITCH' is necessary to get the points to run from right to left.

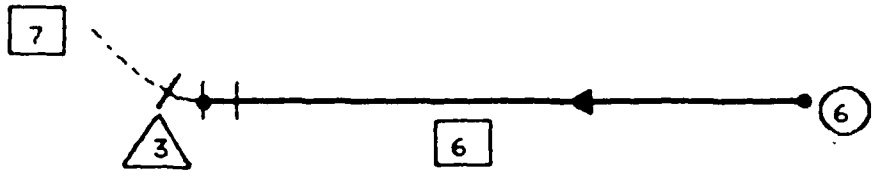


Figure 40. 'SWITCH' for Points on Bottom Outer Boundary

PLACE POINTS ON SEGMENT #8

```
527.  $INPUT ITEM='GETEND', COREIN=7, POINT='LAST', END='FIRST' $
528.  $INPUT ITEM='LINE', POINTS=-8, R2=5, SPACE=7, END='LAST', COREOUT=8 $
```

'GETEND' takes the last point from SETMENT #7 as R1 for 'LINE'. 'LINE'

uses the spacing at the last end of SEGMENT #7 as the spacing at the first end of 'LINE' and uses the number of points set for SEGMENT #8. Line starts at the last point on SEGMENT #7, retrieved by GETEND, and ends at POINT #5.

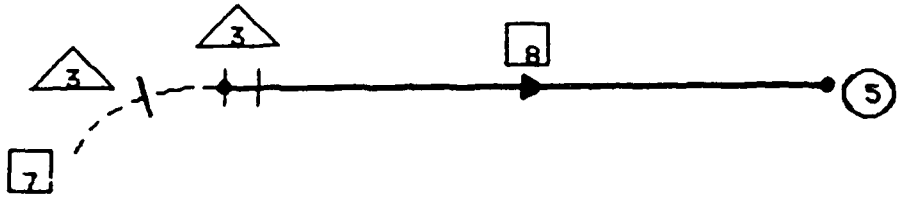


Figure 41. Top Outer Boundary

COMBINE SEGMENTS #6, #7, AND #8 TO FORM #5 (computational)

```
529.  $INPUT ITEM='CURRENT , COREIN=6$
530.  $INPUT ITEM='INSERT', COREIN=7$
531.  $INPUT ITEM='INSERT', COREIN=8, COREOUT=5 $
```

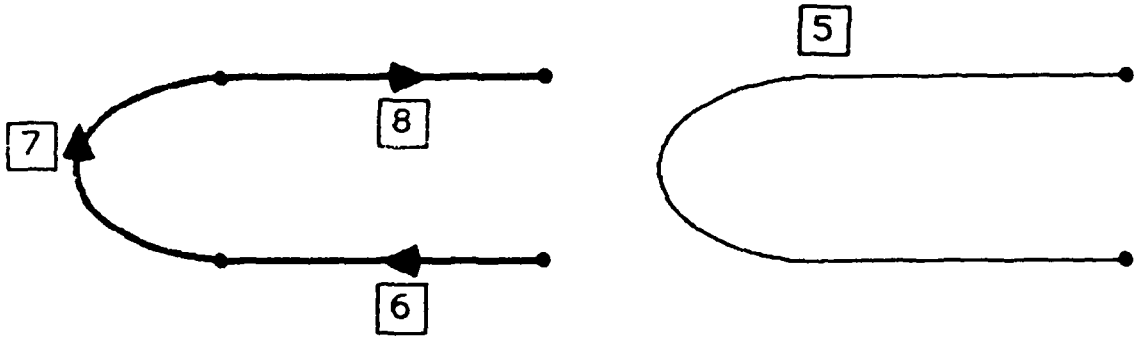


Figure 42. Combining Segments to Form Complete Outer Boundary

PLACE POINTS ON SEGMENT #2

```
532.  $INPUT ITEM='GETEND', COREIN=1, POINT='FIRST', END='FIRST' $
533.  $INPUT ITEM='LINE', POINTS=-2, R2=1, SPACE=-2, RELATIV='NO' $
534.  $INPUT ITEM='SWITCH', REORDER='REVERSE1', COREOUT=2 $
```

'GETEND' takes the point from the trailing edge of the airfoil as R1 for 'LINE'.

'LINE' uses the spacing from SETVAL 2 and the number of points from SETNUM #2. 'LINE' starts at the airfoil trailing edge and ends at POINT #1.

'SWITCH' gets the points to run from right to left.

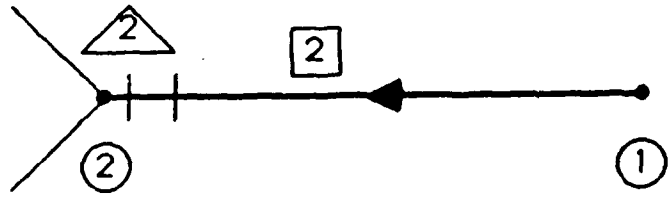


Figure 43. 'SWITCH' to Get Points Oriented Correctly on the Wake Line

```
PLACE POINTS ON SEGMENTS #3 AND #4

535.  $INPUT ITEM='LINE', POINTS=-3, R1=1, R2=6, SPACE=-1, COREOUT=3,
             DISTYP='TANH', RELATIV='NO' $
536.  $INPUT ITEM='LINE', POINTS=-3, R1=1, R2=5, SPACE=-1, COREOUT=4,
             DISTYP='TANH', RELATIV='NO' $
```

Same number of points and same spacing on each.

```
WRITE FILE FOR GRID CODE

537.  $INPUT ITEM='COMBINE', CONTENT='YES', COREIN=1, -5, FILEOUT=1$
538.  $INPUT ITEM='END' $
```

Note that COREIN = 1, -5 reads Files 1, 2, 3, 4, and 5 from temporary memory storage. Combined file stored on File 11.

```
$INPUT ITEM='POINT', POINT=1, R=30.,0. $
$INPUT ITEM='POINT', POINT=6, R=30.,-30. $
$INPUT ITEM='POINT', POINT=5, R=30.,30. $
$INPUT ITEM='SETNUM', SEGMENT=9, POINTS=81 $
$INPUT ITEM='SETNUM', SEGMENT=1, ITERMS= -9,-9 $
$INPUT ITEM='SETNUM', SEGMENT=2, POINTS=31 $
$INPUT ITEM='SETNUM', SEGMENT=3, POINTS=20 $
$INPUT ITEM='SETNUM', SEGMENT=7, POINTS=141 $
$INPUT ITEM='SETNUM', SEGMENT=6, POINTS=41 $
$INPUT ITEM='SETNUM', SEGMENT=5, ITERMS= -2,-1,-2 $
$INPUT ITEM='SETNUM', SEGMENT=8, ITERMS= -5,-6, MATH= 'DIF+1' $
$INPUT ITEM='SETNUM', SEGMENT=8, ITERMS= -8,-7, MATH= 'DIF+1' $
$INPUT ITEM='SETVAL', NUMBER=1, VALUE=.001 $
$INPUT ITEM='SETVAL', NUMBER=2, VALUE=.01 $
$INPUT ITEM='SETVAL', NUMBER=3, VALUE=.0001 $
$INPUT ITEM='SETVAL', NUMBER=4, VALUE=.0015 $
$INPUT ITEM='CURRENT',POINTS= 500     , VALUES=
 0.0000000E+00,  0.0000000E+00,  0.0000000E+00,
  1.000000     ,  0.0000000E+00,  0.0000000E+00  $
$INPUT ITEM='CURDIST', POINTS=-9, SPACE=-4,-2, END= 'LAST', 'LAST',
       COREOUT=9 $
$INPUT ITEM='SWITCH', REORDER='REVERSE1' $
$INPUT ITEM='SCALE', SCALE=1,-1,1 $
$INPUT ITEM='INSERT', COREIN=9, COREOUT=1 $
$INPUT ITEM='CONICUR', TYPE='CIRCLE', RADIUS= 30., ANGLE=270,90,
       POINTS=100, COREOUT=7 $
$INPUT ITEM='CURDIST', COREIN=7, POINTS=-7, SPACE=-3,-3, COREOUT=7 $
$INPUT ITEM='GETEND', COREIN=7, POINT='FIRST', END='FIRST' $
$INPUT ITEM='LINE', POINTS=-6, R2=6, SPACE=7, END='FIRST' $
$INPUT ITEM='SWITCH', REORDER='REVERSE1', COREOUT=6 $
$INPUT ITEM='GETEND', COREIN=7, POINT='LAST', END='FIRST' $
$INPUT ITEM='LINE', POINTS=-8, R2=5, SPACE=7, END='LAST', COREOUT=8 $
$INPUT ITEM='CURRENT',COREIN=6$
$INPUT ITEM='INSERT', COREIN=7$
$INPUT ITEM='INSERT', COREIN=8, COREOUT=5 $
$INPUT ITEM='GETEND', COREIN=1, POINT='FIRST', END='FIRST' $
$INPUT ITEM='LINE', POINTS=-2, R2=1, SPACE=-2, RELATIV='NO' $
$INPUT ITEM='SWITCH', REORDER='REVERSE1', COREOUT=2 $
$INPUT ITEM='LINE', POINTS=-3, R1=1, R2=6, SPACE=-1, COREOUT=3,
           DISTYP='TANH', RELATIV='NO' $
$INPUT ITEM='LINE', POINTS=-3, R1=1, R2=5, SPACE=-1, COREOUT=4,
           DISTYP='TANH', RELATIV='NO' $
$INPUT ITEM='COMBINE', CONTENT='YES', COREIN=1,-5, FILEOUT=1$
$INPUT ITEM='END' $
```

Grid Code Inputs

SET UP BLOCK

```
1.   $INPUT ITEM='INITIAL', KSTORE='CORE', ACCPAR='OPTIMUM', ITMAX=100,
           TOL=1.0E-06, CONTYP='INITIAL' $
2.   $INPUT ITEM='STORE', FILE=69, OUTER='NO' $
3.   $INPUT ITEM='BLOCK', FILE=11 $
```

'INITIAL' with ACCPAR='OPTIMUM' invokes the elliptic grid generation system which uses an algebraic grid to start with.

'STORE' directs that the final grid be placed on File 69.

'BLOCK' directs that all segments will be read from File 11 for this 1 block case.

SET POINT INDICES - SEE COMPUTATIONAL SPACE DIAGRAM

```
4.   $INPUT ITEM='POINT', POINT=1, LOCAT=1,1 $
```

Point #1's cartesian coordinates are associated with I=1, J=1 in curvilinear coordinates at the lower left corner of the computational domain.

```
5.   $INPUT ITEM='POINT', POINT=2, OPOINT=1, SEGMENT=2, DIRECT=1 $
```

Locates Point #2 from #1 using the number of points on SEGMENT #2. DIRECT tells it to go in the direction of increasing values from the first curvilinear coordinate, I.

```
6.   $INPUT ITEM='POINT', POINT=3, OPOINT=2, SEGMENT=1, DIRECT=1 $
7.   $INPUT ITEM='POINT', POINT=4, OPOINT=3, SEGMENT=2, DIRECT=1 $
8.   $INPUT ITEM='POINT', POINT=5, OPOINT=4, SEGMENT=4, DIRECT=2 $
9.   $INPUT ITEM='POINT', POINT=6, OPOINT=5, SEGMENT=5, DIRECT=-1 $
```

(This last one could have been:

```
     $INPUT ITEM='POINT', POINT=6, OPOINT=1, SEGMENT=3, DIRECT=2 $)
```

Notice also that DIRECT=2 tells it to go along the second curvilinear coordinate, J.

SET BLOCK SIZE

```
10.  $INPUT ITEM='SIZE', SIZE=5 $
```

Point #5 determines block size because it contains the maximum computational space values (curvilinear coordinates). That is, at Point #5, I=NI, J=NJ.

SET BOUNDARY VALUES

```
11.  $INPUT ITEM='SEGMENT', SEGMENT= 2, START= 1, END= 2 $
```

'SEGMENT' sets values from Point #1 to Point #2.

```
12.  $INPUT ITEM='SEGMENT', SEGMENT= 2, START= 4, END= 3 $
```

Note the progression from Point #4 to Point #3. Spacing at 3 now matches spacing at 2.

```
13.  $INPUT ITEM='SEGMENT', SEGMENT= 1, START= 2, END= 3 $
14.  $INPUT ITEM='SEGMENT', SEGMENT= 4, START= 4, END= 5, CLASS='FIX' $
15.  $INPUT ITEM='SEGMENT', SEGMENT= 3, START= 1, END= 6, CLASS='FIX' $
16.  $INPUT ITEM='SEGMENT', SEGMENT= 5, START= 6, END= 5, CLASS='FIX' $
```

CLASS='FIX' prevents the points on the far field boundaries from moving along the boundary. That is, they are fixed at the spacings they were set up under in the boundary code.

```
17.  $INPUT ITEM='ORTHOG', START=31,1,1, END=191,1,1, CONUPI=2 $
```

'ORTHOG' invokes a routine that attempts to keep the grid lines off the surface of the airfoil orthogonal. CONUPI controls the frequency of updating the control functions. See ORTHOG in the manual for additional details.

```
18.  $INPUT ITEM='CUT', START=  1, END=  2,
                        ISTART= 4, IEND= 3 $
```

Set up the wake line as a branch cut. See manual for details on branch cuts.

```
19.  $INPUT ITEM='END', CHECK='NO' $
20.  $OUTPUT ITEM='ERROR', BLKERR='YES' $
21.  $OUTPUT ITEM='END' $
```

ITEM='ERROR' asks for residual errors at each iteration. BLKERR='YES' tells it to do ITEM='ERROR' for all blocks.

Note the INPUT and OUTPUT ITEM='END'.

```
$INPUT ITEM='INITIAL', KSTORE='CORE', ACCPAR='OPTIMUM', ITMAX=100,
      TOL=1.0E-06, CONTYP='INITIAL' $
$INPUT ITEM='STORE', FILE=69, OUTER='NO' $
$INPUT ITEM='BLOCK', FILE=11 $
$INPUT ITEM='POINT', POINT=1, LOCAT=1,1 $
$INPUT ITEM='POINT', POINT=2, OPOINT=1, SEGMENT=2, DIRECT=1 $
$INPUT ITEM='POINT', POINT=3, OPOINT=2, SEGMENT=1, DIRECT=1 $
$INPUT ITEM='POINT', POINT=4, OPOINT=3, SEGMENT=2, DIRECT=1 $
$INPUT ITEM='POINT', POINT=5, OPOINT=4, SEGMENT=4, DIRECT=2 $
$INPUT ITEM='POINT', POINT=6, OPOINT=5, SEGMENT=5, DIRECT=-1 $
$INPUT ITEM='SIZE', SIZE=5 $
$INPUT ITEM='SEGMENT', SEGMENT= 2, START= 1, END= 2 $
$INPUT ITEM='SEGMENT', SEGMENT= 2, START= 4, END= 3 $
$INPUT ITEM='SEGMENT', SEGMENT= 1, START= 2, END= 3 $
$INPUT ITEM='SEGMENT', SEGMENT= 4, START= 4, END= 5, CLASS='FIX' $
$INPUT ITEM='SEGMENT', SEGMENT= 3, START= 1, END= 6, CLASS='FIX' $
$INPUT ITEM='SEGMENT', SEGMENT= 5, START= 6, END= 5, CLASS='FIX' $
$INPUT ITEM='ORTHOG', START=31,1,1, END=191,1,1, CONUPI=2 $
$INPUT ITEM='CUT', START=  1, END=  2,
                   ISTART= 4, IEND= 3 $
$INPUT ITEM='END', CHECK='NO'  $
$OUTPUT ITEM='ERROR', BLKERR='YES' $
$OUTPUT ITEM='END' $
```

CONE-CYLINDER-FLARE (3-D EXAMPLE)

This example will illustrate the procedure for building a three-dimensional grid composed of two blocks to describe the flow field around a cone-cylinder-flare body. A line-by-line description of the inputs to the boundary code and the inputs to the grid code will be presented in addition to graphical illustrations showing the result of each line. Figures 44 and 45 are line drawings of the physical space and the computational space. The computational domain is required for the solution of the flow field using finite difference or finite volume techniques. Two blocks, each having dimensions of 65 x 18 x 31 are combined to form the total region of interest around the cone-cylinder-flare body. Symmetry conditions could be imposed and as a result only one block would be necessary. The overall dimensions which describe the flow field were selected to make use of the EAGLE Flow Solver. Specifically, the dimensions were selected to determine the flow field around a cone-cylinder-flare body operating at Mach = 2.81 and various angles of attack.

Finally, please note that the ITEM="POINT" commands were not used in the grid input data. An alternate method of grid generation is used for this example as compared to the NACA 0012 example, presented previously.

BOUNDARY GENERATION

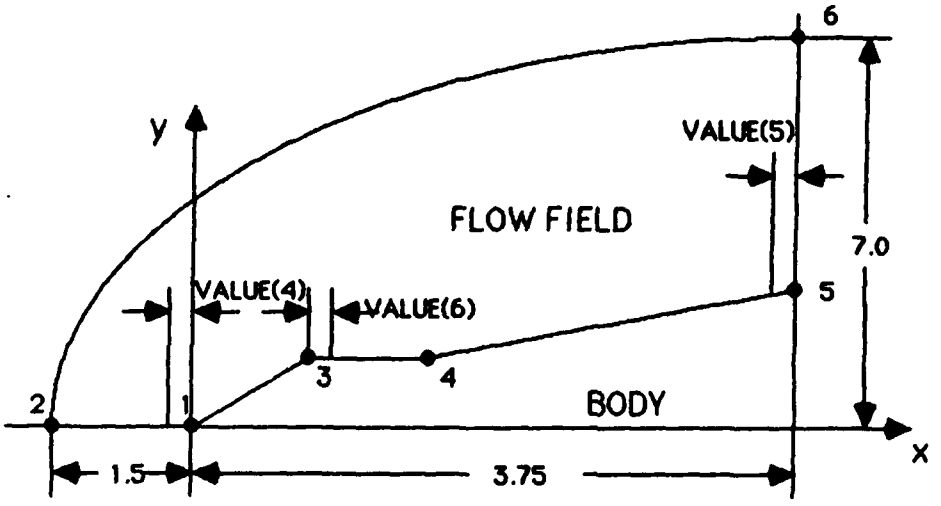


Figure 44. Flow Field in Physical Space

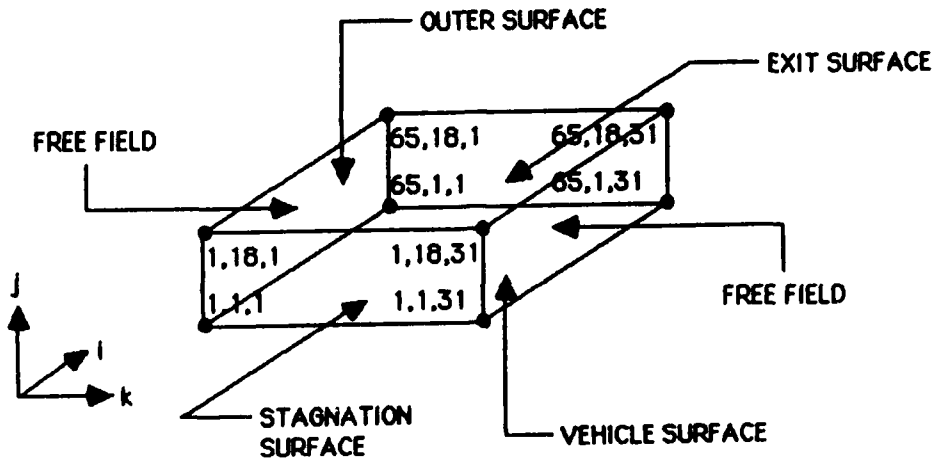


Figure 45. Computational Space (Block 1 and Block 2)

SET GEOMETRICAL VALUES

1. $INPUT ITEM="SETVAL",NUMBER=4,VALUE=.005$
2. $INPUT ITEM="SETVAL",NUMBER=5,VALUE=.005$
3. $INPUT ITEM="SETVAL",NUMBER=6,VALUE=.039$

These commands specify the values of the "SPACE=" input variables.

DEFINE END POINTS OF LINE SEGMENTS

4. $INPUT ITEM="POINT",POINT=1,R=0.0,0.0,0.0$
5. $INPUT ITEM="POINT",POINT=2,R=-1.50,0.0,0.0$
6. $INPUT ITEM="POINT",POINT=3,R=.90,.375,0.0$
7. $INPUT ITEM="POINT",POINT=4,R=1.65,.375,0.0$
8. $INPUT ITEM="POINT",POINT=5,R=3.75,.750,0.0$
9. $INPUT ITEM="POINT",POINT=6,R=3.75,7.00,0.0$

The point command specifies the end-points of a line segment which will outline the region in which grid generation will be performed. These commands are used by specifying R1= "POINT NO." and R2= "POINT NO.", where R1 and R2 are the x, y, and z coordinates of the first and second end points of a line segment. These commands appear where "LINE" segment input data are used to specify the shape of the vehicle surface. A subsequent rotation produces a three-dimensional grid field.

OUTER BOUNDARY (Block 1)

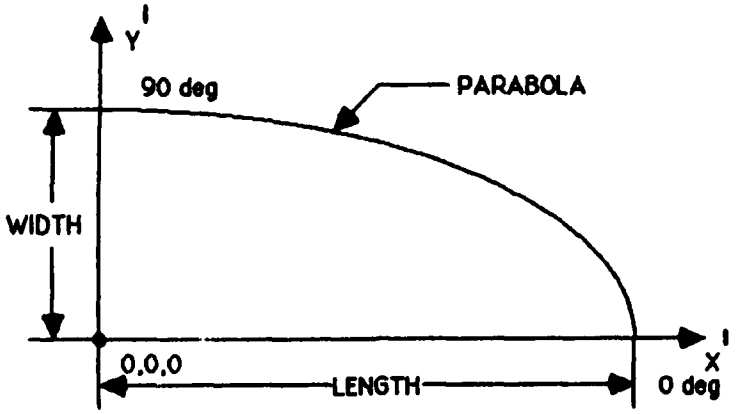


Figure 46. Curve Describing Outer Surface

10. $INPUT ITEM="CONICUR",CURPTS=100,TYPE="PARABOLA",LENGTH=5.250,
WIDTH=7.00,ANGLE=0,90$

Develops a conic boundary in the x', y', z' coordinate system. A parabola is generated which has 100 points and is specified by a length of 5.250 and a width of 7.00. The curve is generated in the x', y', z', system by developing the parabola from the first point at 0^0 to the second point at 90^o.

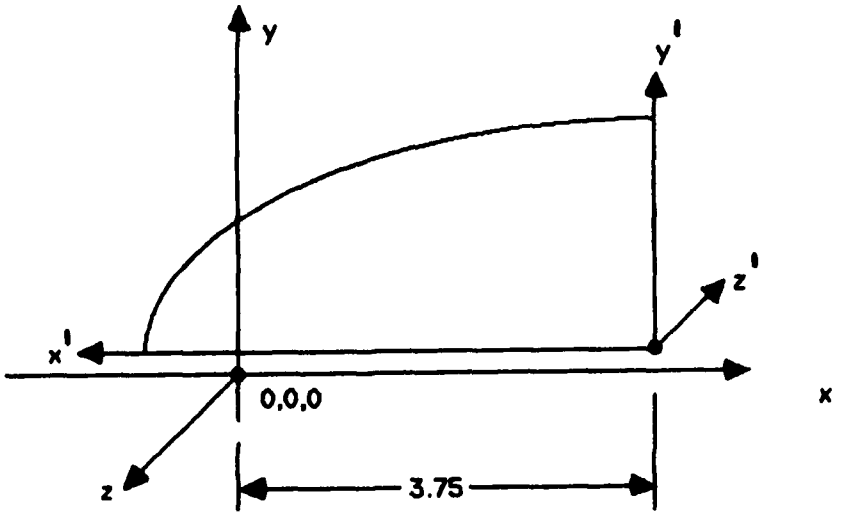


Figure 47. Curve of Outer Surface Transformed to Global Coordinates

11. $INPUT ITEM="TRANS",POINTS=100,ORIGIN=3.750,0,0,COSINES=-1,0,0,0,1,0,
0,0,-1$

This command performs a coordinate transformation of the form, $X' = a_{ij}X + T(x,y,z)$. The local coordinate system (x', y', z') in which the curve is specified is transformed relative to the global system (x,y,z) by using the transformation matrix, a_{ij} and a translation term, T(x,y,z). This process transforms the previous parabolic arc to form the outer boundary of the solution region. The origin of the transformed axes (x', y', z') is placed x = 3.75, y = 0, z = 0 from the origin of the basic coordinate system. One hundred points are transformed by this command.

12. $INPUT ITEM="CURDIST",POINTSI=100,DISTYP="LINEAR",POINTS=65,COREOUT=11$

Distributes 65 points on the curve that had 100 points. A linear distribution is used. The curve is stored in core location number 11.

13. $INPUT ITEM="BOUNCUR",POINTS=65$

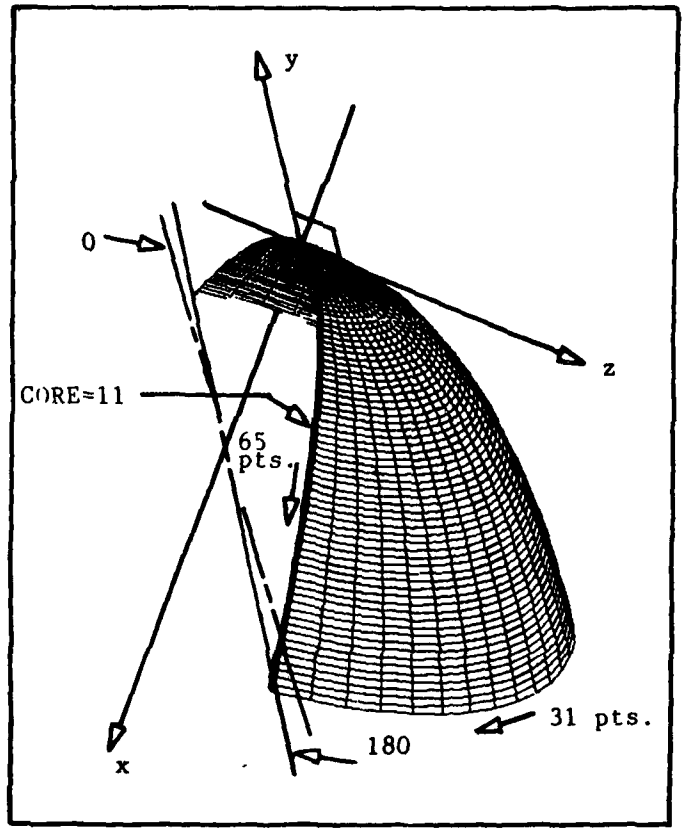


Figure 48. Boundary of Outer Surface of Solution Region

14. $INPUT ITEM="ROTATE",CURPTS=65,ANGPTS=31,ANGLE=0,180,AXCOS=1,0,0
NORCOS=0,1,0,COREOUT=5$

Generates a surface using a 65-point curve which represents block 1 of the outer surface. The curve is placed at 31 angular positions to generate the surface. The results are stored into core location number 5.

STAGNATION LINE (Block 1, Block 2)

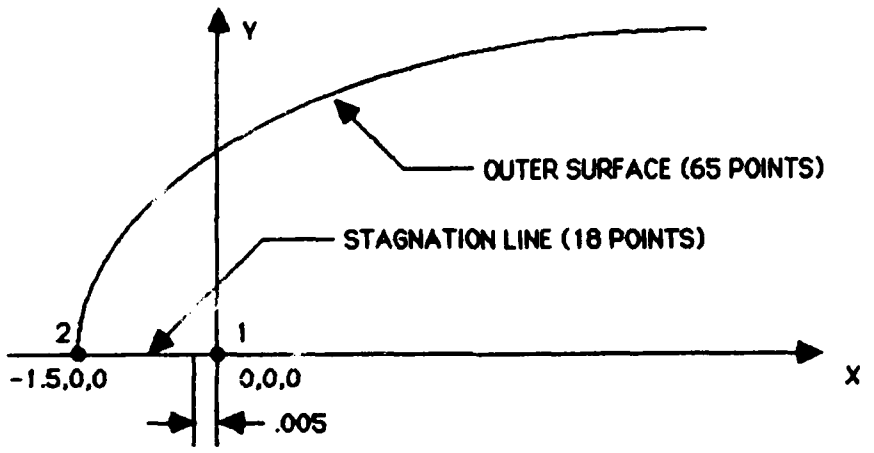


Figure 49. Stagnation Line Definition

15. $INPUT ITEM="LINE",CURPTS=18,R1=1,R2=2,DISTYP="TANH",
SPACE=-4,RELATIV="NO",COREOUT=20$

Creates a line from point 1 (0,0,0) to point 2 (-1.50,0,0) using 18 points. A hyperbolic tangent distribution is used with the first point off the nose (0,0,0) a distance of .005 (SPACE = -4). The data is stored in core location number 20.

16. $INPUT ITEM="BOUNCUR",COREIN=20,POINTS=18$

The stagnation line is used as an axis to perform a rotation. The stagnation line is used as a bounding curve with the prescribed number of points (CURPTS=18) used to generate a stagnation surface.

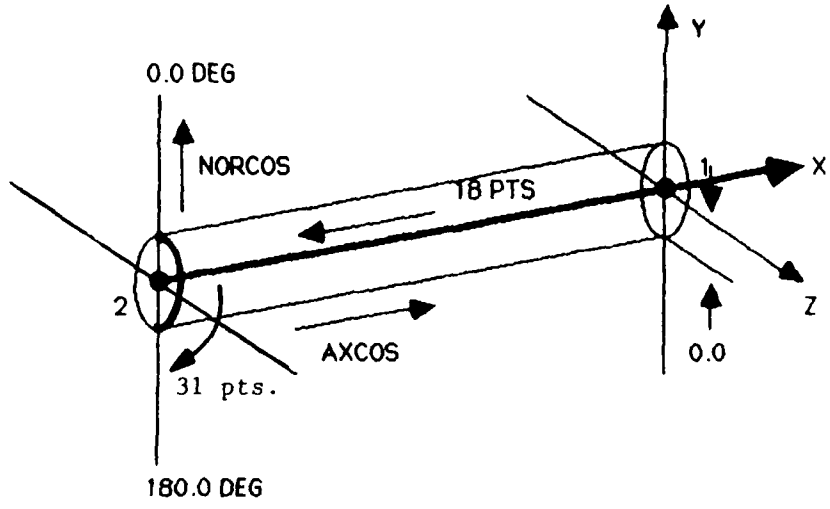


Figure 50. Block 1, Stagnation Surface

17. $INPUT ITEM="ROTATE",CURPTS=18,ANGPTS=31,ANGLE=0,180,
AXCOS=1,0,0,NORCOS= 0,1,0,COREOUT=1$

A stagnation line having 31 coincident angular-points at each point in the -X direction is generated. This is a stagnation surface from which connectivity to the outer surface is now made possible. This block 1 data is stored in core location number 1.

18. $INPUT ITEM="BOUNCUR",COREIN=20,POINTS=18$

The stagnation line is input from core location number 20 and will be used as an axis to perform a rotation to create a surface.

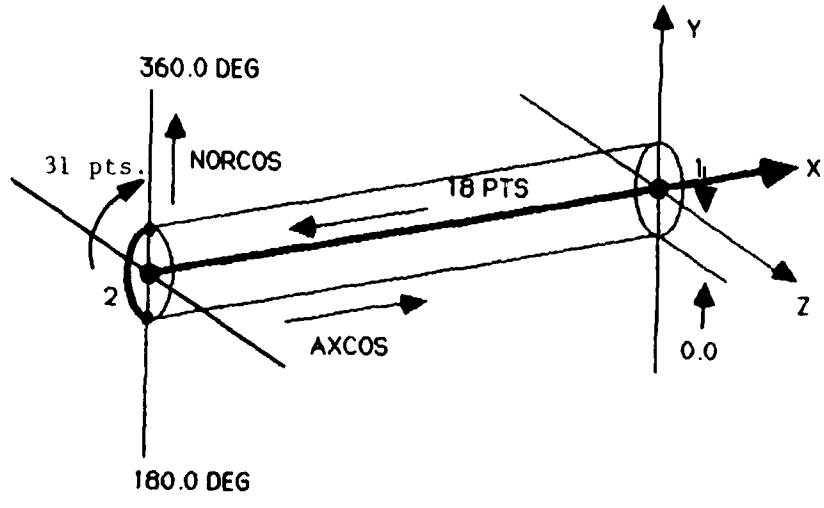


Figure 51. Block 2, Stagnation Surface

19. $INPUT ITEM="ROTATE",CURPTS=18,ANGPTS=31,ANGLE=180,360
AXCOS=1,0,0,NORCOS=0,1,0,COREOUT=2$

A stagnation line having 31 coincident angular-points at each point in the -X direction is generated. This is a stagnation surface from which connectivity to the outer surface is now made possible. This block 2 data is stored in core location number 2.

20. $INPUT ITEM="COMBINE",COREIN=1,2,FILEOUT=2$

The stagnation surfaces for block 1 and block 2 are combined onto one file and stored in file number 2.

BACK SURFACE (Block 1, Block 2)

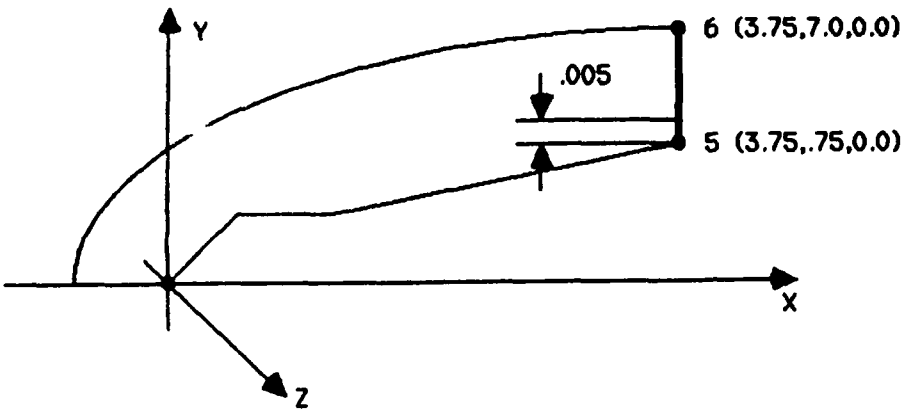


Figure 52. Exit-Plane Definition

21. $INPUT ITEM="LINE",CURPTS=18,R1=5,R2=6,DISTYP="TANH",
SPACE=-5,RELATIV="NO",COREOUT=21$

This command creates a line from point 5 (3.75, .75, 0) to point 6 (3.75, 7.0, 0) with 18 points between these values. A hyperbolic tangent function is used for point distribution, the second point is located .005 from the surface of the body in the Y direction. This file is written to core location number 21.

22. $INPUT ITEM="BOUNCUR",COREIN=21,POINTS=18$

The line input from core location number 21 is used to create a surface by a rotation that will follow. This command defines the line as an axis of rotation for the creation of a surface.

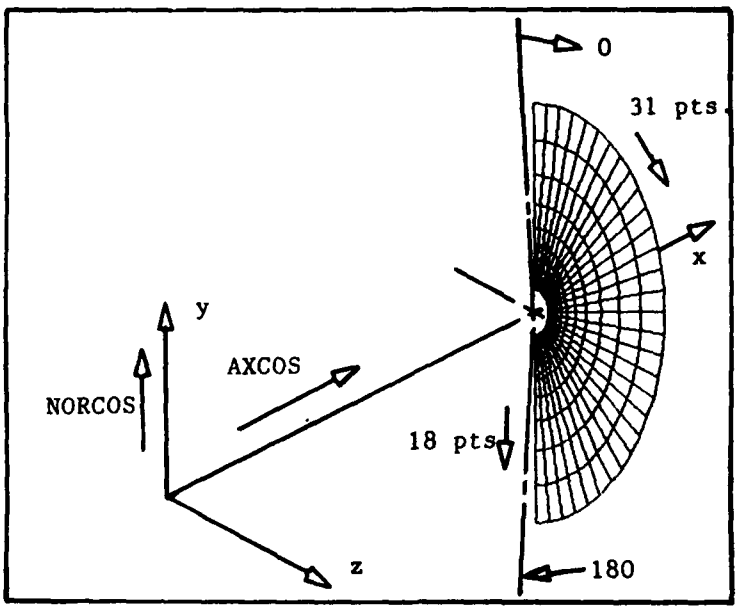


Figure 53. Block 1, Exit-Boundary

23. $INPUT ITEM="ROTATE,CURPTS=18,ANGPTS=31,ANGLE=0,180,
AXCOS=1,0,0,NORCOS=0,1,0,COREOUT=3$

This command generates the exit-flow boundary surface for block 1 by rotating the line that is input from core location number 21. A surface measuring 18 x 31 points is generated to form the exit-flow boundary surface. The results are stored in core location number 3. The axial and normal direction unit vectors are given by AXCOS=1,0,0 and NORCOS=0,1,0 respectively.

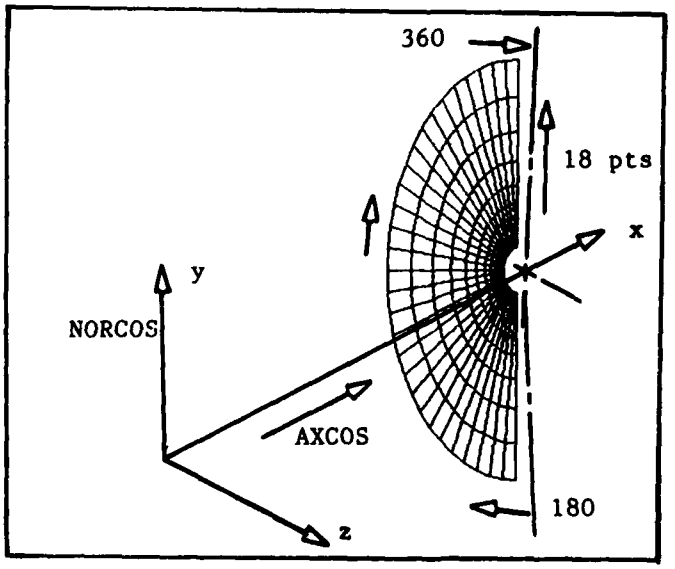


Figure 54. Block 2, Exit Boundary

24. $INPUT ITEM="BOUNCUR",COREIN=21,POINTS=18$

The line input from core location number 21 will be used to create a surface by a rotation that will follow. This command defines the line as an axis of rotation for the creation of a surface.

25. $INPUT ITEM="ROTATE",CURPTS=18,ANGPTS=31,ANGLE=180,360,
AXCOS=1,0,0 NORCOS=0,1,0COREOUT=4$

This command generates the exit-flow boundary surface for block 2 by rotating the line that is input from core location number 21. A surface measuring 18 x 31 points is generated to form the exit-flow boundary surface. The results are stored in core location number 4. The axial and normal direction unit vectors are given by AXCOS=1,0,0 and NORCOS=0,1,0, respectively.

26. $INPUT ITEM="COMBINE",COREIN=3,4,FILEOUT=3$

The exit-plane formed from the block 1 and the block 2 grid data are combined onto one file and stored in file number 3. This forms the total exit surface boundary of the solution region.

INNER BOUNDARY (Block 1)

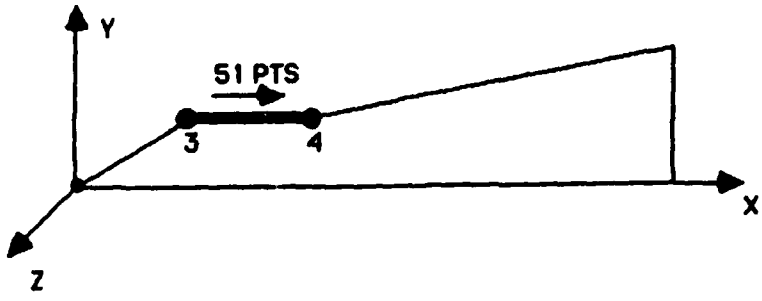


Figure 55. Line Segment 1 of Vehicle Boundary

27. $INPUT ITEM="LINE",CURPTS=51,R1=3,R2=4,DISTYP="LINEAR",
RELATIV="NO",COREOUT=13$

This input develops a straight line with 51 points from point 3 (.9, .375, 0) to point 4 (1.65, .375, 0). A linear distribution is used to distribute 51 points along the length of the line. The parameter DISTYP="LINEAR" selects a Linear function to distribute the points. The points are then written to core location number 13 (COREOUT=13).

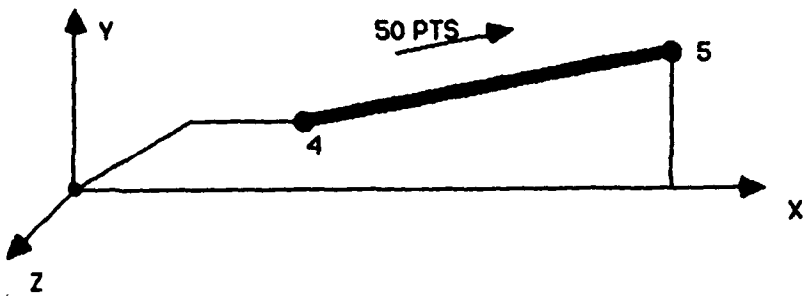


Figure 56. Line 2 Segment of Vehicle Boundary

28. $INPUT ITEM="LINE",CURPTS=50,R1=4,R2=5,DISTYP="LINEAR",
RELATIV="NO",COREOUT=14$

This input develops a straight line with 50 points from point 4 (1.65, .375, 0) to point 5 (3.75, .75, 0). A linear distribution is used to distribute 50 points along the length of the line. The parameter DISTYP="LINEAR", selects a linear function to distribute the points. The points are then written to core location number 14 (COREOUT=14).

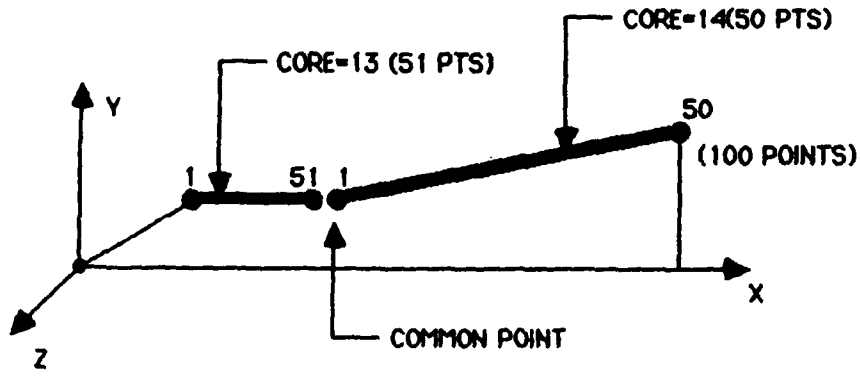


Figure 57. Insertion of Line Segment 2 to Segment 1

29. $INPUT ITEM="CURRENT",COREIN=13,POINTS=51$

This input takes the 51 points from core location number 13 and puts it into the current position in central memory.

30. $INPUT ITEM = "INSERT", COREIN = 14, POINTSI = 50, START = 51, POINTS = 100$

This input inserts the 50 points stored in temporary core location number 14 to the end of the line which is point 51 on the current curve. POINTS=50 describes the number of points being inserted. START = 51, indicates which point on the current curve to begin insertion of the 50 point curve from core location number 14. A total of 100 points are generated (POINTS=100).

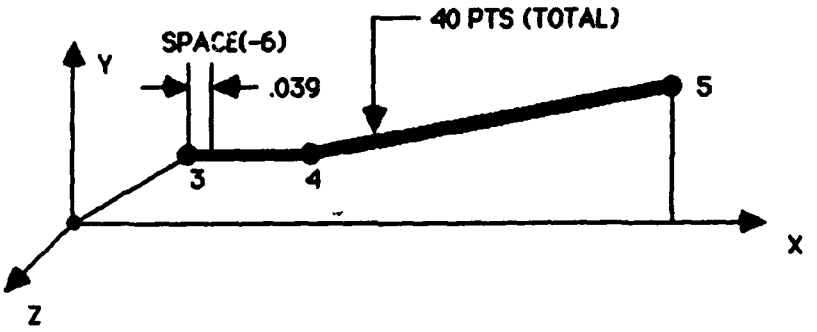


Figure 58. Point Redistribution of Line Segment 3

31. $INPUT ITEM="CURDIST",POINTSI=100,DISTYP="TANH",POINTS=40,
SPACE=-6,RELATIV="NO",COREOUT=14$

This command will redistribute the 100 point curve (POINTSI=100) and use a hyperbolic tangent function (DISTYP="TANH") to space the points from the first point (R1 = 3) to the last point (R2 = 5). The second point off point 3 is located a distance .039 from that point (SPACE=-6). The spacings will be taken as actual arc lengths because RELATIV="NO" is specified. Finally, the 40 point curve is saved in temporary core location, number 14. (COREOUT=14).

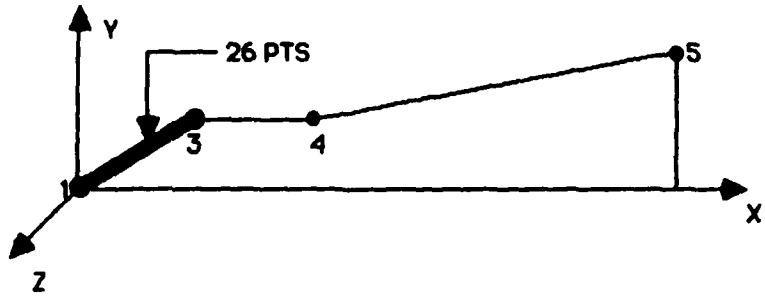


Figure 59. Line Segment 4 of Vehicle Boundary

32. $INPUT ITEM="LINE",CURPTS=26,R1=1,R2=3,DISTYP="LINEAR",
RELATIV="NO"$

This input develops a straight line with 26 points from point 1 (0,0,0) to point 3 (.90, .375, 0.0). A linear distribution is used to space the 26 points along the length of the line. The parameter, DISTYP="LINEAR" selects a linear function to distribute the points. The points are then available in the current position in memory for further processing.

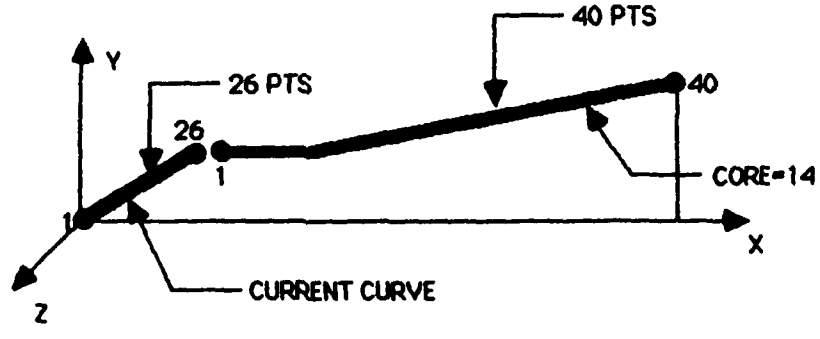


Figure 60. Insertion of Line Segment 3 to End of Line Segment 4

33. $INPUT ITEM="INSERT",COREIN=14,POINTSI=40,START=26,POINTS=65,
COREOUT=12$

This input inserts the 40 points stored in temporary core location number 14 to the end of the line which is point 26 on the current curve. POINTS=40 describes the number of points being inserted. START=51 indicates which point on the current curve to begin insertion of the 40 point curve from core location number 14. A total of 65 points are generated (POINTS=65). The line, which describes the shape of the body, is output to memory location, COREOUT=12.

34. $INPUT ITEM="BOUNCUR",POINTS=65$

This input sets the current curve as a bounding curve with the prescribed number of points (POINTS=65). This will allow the rotation of the curve to form a surface.

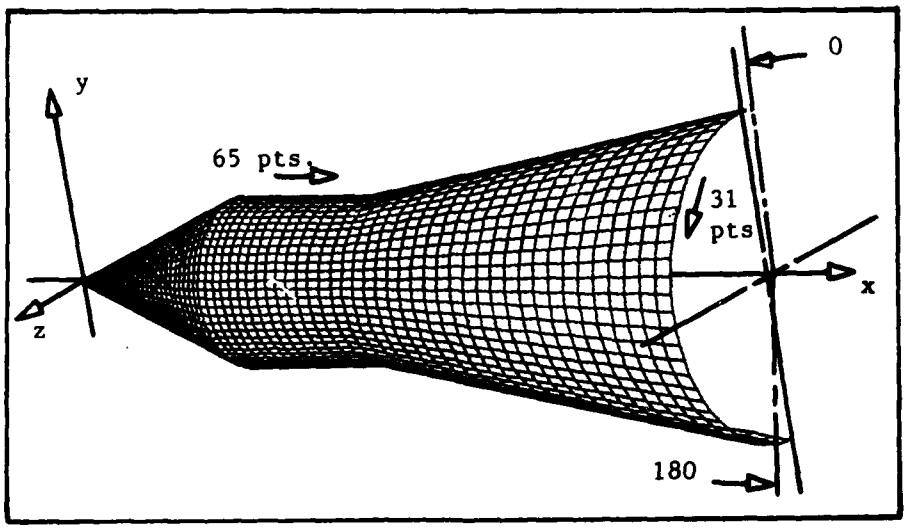


Figure 61. Block 1, Vehicle Boundary

35. $INPUT ITEM="ROTATE",CURPTS=65,ANGPTS=31,ANGLE=0,180,AXCOS=1,0,0
NORCOS=0,1,0,COREOUT=6$

Generates a surface using a 65 point curve (CURPTS=65) which represents

block 1 of the vehicle surface. The curve is placed at 31 angular positions to generate the surface. The results are stored into core location number 6.

INNER/OUTER SURFACE (Block 1)

36. $INPUT ITEM="COMBINE",COREIN=5,6,FILEOUT=4$

The files describing the surfaces of the outer boundary of the solution region and the inner boundary or vehicle surface are put onto file number 4.

OUTER BOUNDARY (Block 2)

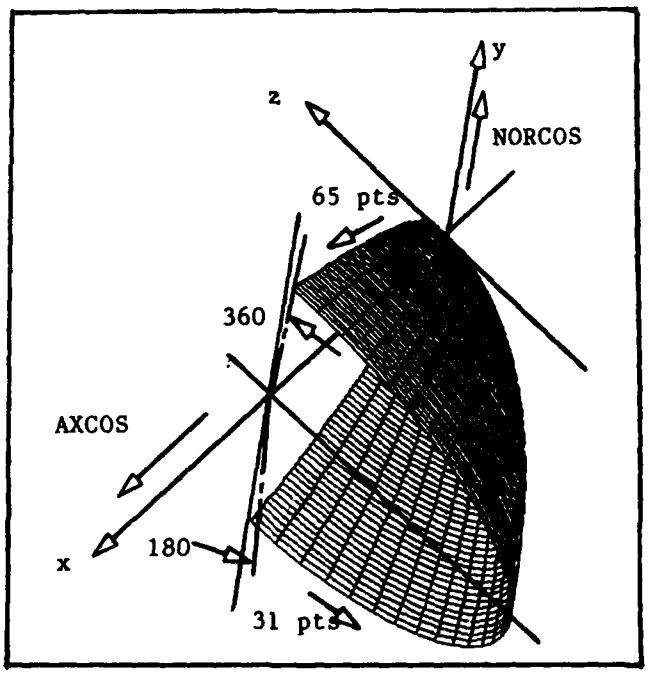


Figure 62. Block 2, Boundary of Outer Surface of Solution Region

37. $INPUT ITEM="CURRENT",COREIN=11$

The curve (parabola) which describes the curvature of the outer surface of

the flow region is made current in memory.

38. $INPUT ITEM="BOUNCUR",POINTS=65$

The curve of the outer surface is defined as an axis of rotation for the generation of an outer surface for block number 2.

39. $INPUT ITEM="ROTATE",CURPTS=65,ANGPTS=31,ANGLE=180,360,AXCOS=1,0,0,
NORCOS=0,1,0,COREOUT=8$

Generates a surface using the 65 point curve which describes the axisymmetric shape of the outer surface. The curve is placed at 31 angular positions to generate the block 2 surface of the outer boundary. The surface generation proceeds from the points located at 180^0 to the points located at 360^o. The results are stored into core location number 8.

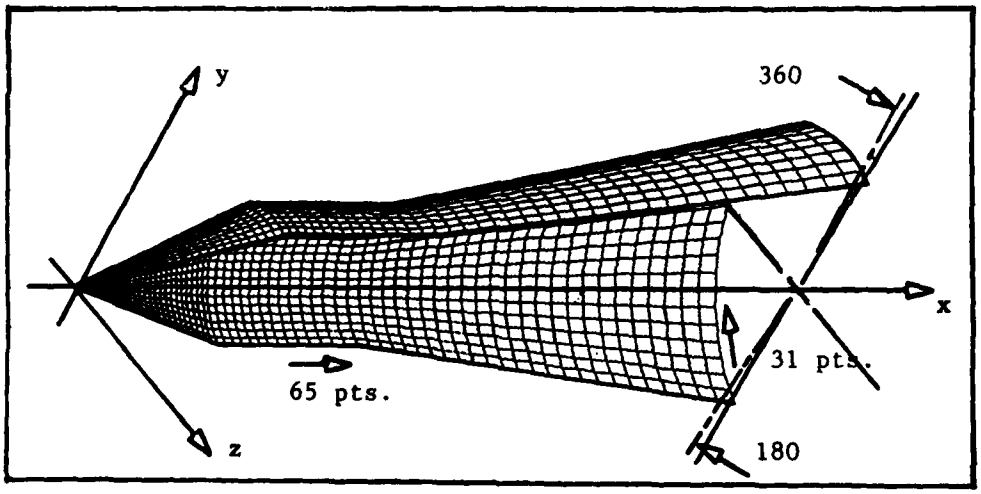


Figure 63. Block 2, Vehicle Boundary

INNER BOUNDARY (Block 2)

40. $INPUT ITEM="CURRENT",COREIN=12$

The composite line which describes the curvature of the vehicle surface is

made current in memory.

41. $INPUT ITEM="BOUNCUR",POINTS=65$

The line which was made current is defined as an axis of rotation for the generation of the vehicle surface for block 2.

42. $INPUT ITEM="ROTATE",CURPTS=65,ANGPTS=31,ANGLE=180,360,AXCOS=1,0,0,
NORCOS=0,1,0,COREOUT=9$

Generates a surface using a 65 point curve. The curve is placed at 31 angular positions to generate the block 2 surface of the vehicle boundary. The surface generation proceeds from the points located at 180^{o} to the points located at 360^{o}. The results are stored into core location number 9.

INNER/OUTER BOUNDARY (Block 2)

43. $INPUT ITEM="COMBINE",COREIN=8,9,FILEOUT=5$

The inner surface from core location number 9 and the outer surface from core location number 5 are combined to create the file describing the inner/outer surfaces for block 2. Results are stored into file number 5.

44. $INPUT ITEM="END"$

```
*.      SET GEOMETRICAL VALUES
SINPUT ITEM="SETVAL",NUMBER=4,VALUE=.005$
SINPUT ITEM="SETVAL",NUMBER=5,VALUE=.005$
SINPUT ITEM="SETVAL",NUMBER=6,VALUE=.039$
*.      DEFINE END POINTS OF LINE SEGMENTS
SINPUT ITEM="POINT",POINT=1,R=0.0,0.0,0.0$
SINPUT ITEM="POINT",POINT=2,R=-1.50,0.0,0.0$
SINPUT ITEM="POINT",POINT=3,R=.90,.375,0.0$
SINPUT ITEM="POINT",POINT=4,R=1.65,.375,0.0$
SINPUT ITEM="POINT",POINT=5,R=3.75,.750,0.0$
SINPUT ITEM="POINT",POINT=6,R=3.75,7.00,0.0$
*.      OUTER BOUNDARY
SINPUT ITEM="CONICUR",CURPTS=100,TYPE="PARABOLA",LENGTH=5.250,
  WIDTH=7.00,ANGLE=0,90$
SINPUT ITEM="TRANS",POINTS=100,ORIGIN=3.750,0,0,COSINES=-1,0,0,0,1,0,
  0,0,-1$
SINPUT ITEM="CURDIST",POINTSI=100,DISTYP="LINEAR",POINTS=65,COREOUT=11$
SINPUT ITEM="BOUNCUR",POINTS=65$
SINPUT ITEM="ROTATE",CURPTS=65,ANGPTS=31,ANGLE=0,180,AXCOS=1,0,0,
  NORCOS=0,1,0,COREOUT=5$
*.      STAGNATION LINE
SINPUT ITEM="LINE",CURPTS=18,R1=1,R2=2,DISTYP="TANH",
  SPACE=-4,RELATIV="NO",COREOUT=20$
SINPUT ITEM="BOUNCUR",COREIN=20,POINTS=18$
SINPUT ITEM="ROTATE",CURPTS=18,ANGPTS=31,ANGLE=0,180,
  AXCOS=1,0,0,NORCOS=0,1,0,COREOUT=1$
SINPUT ITEM="BOUNCUR",COREIN=20,POINTS=18$
SINPUT ITEM="ROTATE",CURPTS=18,ANGPTS=31,ANGLE=180,360,
  AXCOS=1,0,0,NORCOS=0,1,0,COREOUT=2$
SINPUT ITEM="COMBINE",COREIN=1,2,FILEOUT=2$
*.      BACK SURFACE
SINPUT ITEM="LINE",CURPTS=18,R1=5,R2=6,DISTYP="TANH",
  SPACE=-5,RELATIV="NO",COREOUT=21$
SINPUT ITEM="BOUNCUR",COREIN=21,POINTS=18$
SINPUT ITEM="ROTATE",CURPTS=18,ANGPTS=31,ANGLE=0,180,
  AXCOS=1,0,0,NORCOS=0,1,0,COREOUT=3$
SINPUT ITEM="BOUNCUR",COREIN=21,POINTS=18$
SINPUT ITEM="ROTATE",CURPTS=18,ANGPTS=31,ANGLE=180,360,
  AXCOS=1,0,0,NORCOS=0,1,0,COREOUT=4$
SINPUT ITEM="COMBINE",COREIN=3,4,FILEOUT=3$
*.      INNER BOUNDARY - BLOCK1
SINPUT ITEM="LINE",CURPTS=51,R1=3,R2=4,DISTYP="LINEAR",
  RELATIV="NO",COREOUT=13$
SINPUT ITEM="LINE",CURPTS=50,R1=4,R2=5,DISTYP="LINEAR",
  RELATIV="NO",COREOUT=14$
SINPUT ITEM="CURRENT",COREIN=13,POINTS=51$
SINPUT ITEM="INSERT",COREIN=14,POINTSI=50,START=51,POINTS=100$
SINPUT ITEM="CURDIST",POINTSI=100,DISTYP="TANH",POINTS=40,
  SPACE=-6,RELATIV="NO",COREOUT=14$
SINPUT ITEM="LINE",CURPTS=26,R1=1,R2=3,DISTYP="LINEAR",
  RELATIV="NO"$
SINPUT ITEM="INSERT",COREIN=14,POINTSI=40,START=26,POINTS=65,
  COREOUT=12$
SINPUT ITEM="BOUNCUR",POINTS=65$
SINPUT ITEM="ROTATE",CURPTS=65,ANGPTS=31,ANGLE=0,180,AXCOS=1,0,0,
  NORCOS=0,1,0,COREOUT=6$
*.      INNER/OUTER BOUNDARY - BLOCK1
SINPUT ITEM="COMBINE",COREIN=5,6,FILEOUT=4,
  OUT="PLOT",SYMBOL=0,
  RMIN=-1.50,-7.0,-7.0,RMAX=3.75,7.0,7.0$
*.      OUTER BOUNDARY - BLOCK2
SINPUT ITEM="CURRENT",COREIN=11$
SINPUT ITEM="BOUNCUR",POINTS=65$
SINPUT ITEM="ROTATE",CURPTS=65,ANGPTS=31,ANGLE=180,360,AXCOS=1,0,0,
  NORCOS=0,1,0,COREOUT=8$
*.      INNER BOUNDARY - BLOCK2
```

```
$INPUT ITEM="CURRENT",COREIN=12$
$INPUT ITEM="BOUNCUR",POINTS=65$
$INPUT ITEM="ROTATE",CURPTS=65,ANGPTS=31,ANGLE=180,360,AXCOS=1.0,0.,
  NORCOS=0,1,0,COREOUT=9$
*.      INNER/OUTER BOUNDARY - BLOCK2
$INPUT ITEM="COMBINE",COREIN=8,9,FILEOUT=5,
  OUT="PLOT",SYMBOL=0,
  RMIN=-1.50,-7.0,-7.0,RMAX=3.75,7.0,7.0$
$INPUT ITEM="END"$
```

CONE-CYLINDER-FLARE (3-D EXAMPLE)

GRID GENERATION

1. $INPUT ITEM="STORE",FILE=75,KSTORE="CORE",ITMAX=1,TOL=1.E-05,
   TRIAD="YES",FORM="E",ACCPAR="OPTIMUM"$

This command will store the grid coordinates to be generated on file, gf75. The output will occur when the maximum number of iterations (ITMAX = 1) or the tolerance of the residuals have been reached (TOL = 1. E-05). Since this code uses SOR (Successive Over-Relaxation), the acceleration parameter is specified (ACCPAR = OPTIMUM").

2. $INPUT ITEM="INITIAL",BLEND="ARC","ARC",CHECK="NO",OUTER="NO",
   CONTYP="RADIUS",ALL="YES"$

This command initializes the grid generation program. The blending function in all three directions are set to use interpolations based on arc length, BLEND="ARC","ARC"). The execution of the routine will continue even though the jacobian may be zero or negative, (CHECK="NO"). The routine will write out the cartesian coordinates of the grid to separate arrays, (OUTER="SEPARATE"). Finally, the type of control function selected is based on local radius, (CONTYP="RADIUS").

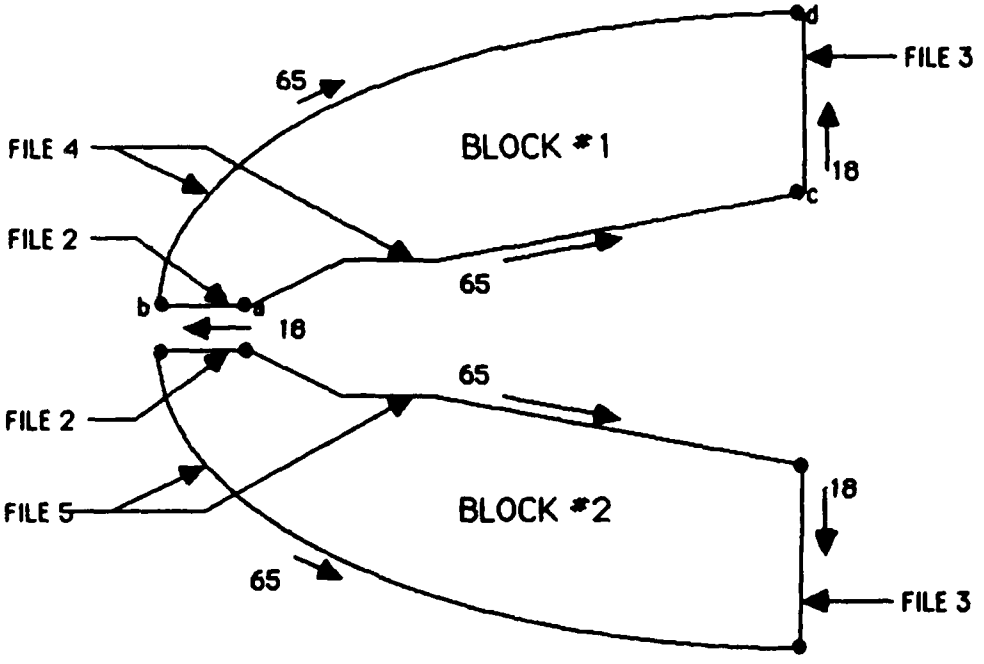


Figure 64. Connectivity of Block 1 to Block 2

3. $INPUT ITEM = "BLOCK",SIZE=65,18,31$

This input line states the size of each block (SIZE = 65, 18, 31).

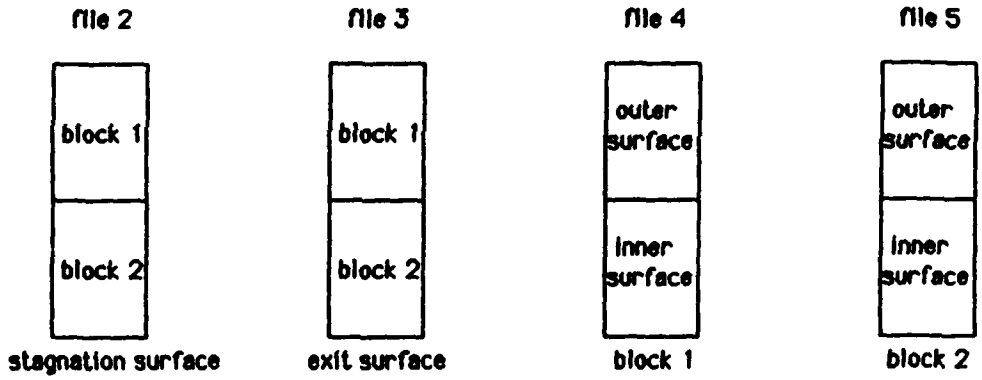


Figure 65. Files Created by Boundary Program and Their Location

4. $INPUT ITEM="FILE",START=1,18,1,END=65,18,31,CLASS="FIX",FILE=14,
REWIND="YES"$

5. $INPUT ITEM="FILE",START=1,1,1,END=65,1,31,CLASS="FIX",FILE=14,
REWIND="NO"$

6. $INPUT ITEM="FILE",START=65,1,1,END=65,18,31,CLASS="FIX",FILE=13,
REWIND="YES"$

7. $INPUT ITEM="FILE",START=1,1,1,END=1,18,31,CLASS="FIX",FILE=12,
REWIND="YES"$

These input lines read in the boundary surfaces generated in the previous surface generation routine. The points are read from a file (FILE=XX) based on the inputs (START=x,y,z,END=x,y,z). These points are also classified as to what type of points they are (CLASS="FIX"). REWIND="YES" rewinds the file prior to reading.

8. $INPUT ITEM="INTERP",START=1,1,1,END=65,18,1,BLEND="ARC","ARC","ARC"$

9. $INPUT ITEM="INTERP",START=1,1,31,END=65,18,31,BLEND="ARC","ARC","ARC"$

These commands determine the unknown points on the sides of the computational block by interpolation. Each section with unknown points is described by START= . . ., END= . . ., and they are also classified for boundary conditions

(CLASS="FIX"). The type of interpolation is given by specifying the interpolation used for the blending function, (BLEND="ARC",ARC","ARC").

BLOCK 2

```
10.  $INPUT ITEM="BLOCK",SIZE=65,18,31$
11.  $INPUT ITEM="FILE",START=1,18,1,END=65,18,31,CLASS="FIX",FILE=15,
       REWIND=YES
12.  $INPUT ITEM="FILE",START=1,1,1,END=65,1,31,CLASS="FIX",FILE=15,
       REWIND="NO"$
13.  $INPUT ITEM="FILE",START=65,1,1,END=65,18,31,CLASS="FIX",FILE=13,
       REWIND="NO"$
14.  $INPUT ITEM="FILE",START=1,1,1,END=1,18,31,CLASS="FIX",FILE=12,
       REWIND="NO"$
```

These input lines read in the boundary surfaces generated in the previous surface generation routine. The points are read from a file (FILE=xx) based on the inputs (START=x,y,z,END=x,y,z). These points are also classified as to what type of points they are (CLASS="FIX"). REWIND="YES" rewinds the file prior to reading.

```
15.  $INPUT ITEM="INTERP",START=1,1,1,END=65,18,1,BLEND="ARC","ARC","ARC"$
16.  $INPUT ITEM = "INTERP"START=1,1,31,END=65,18,31,BLEND="ARC","ARC","ARC"$
```

These commands determine the unknown points on the sides of the computational block by interpolation. Each section with unknown points is described by START=. . ., END=. . ., and they are also classified for boundary conditions (CLASS="FIX"). The type of interpolation is given by specifying the interpolation used for the blending function (BLEND="ARC","ARC","ARC").

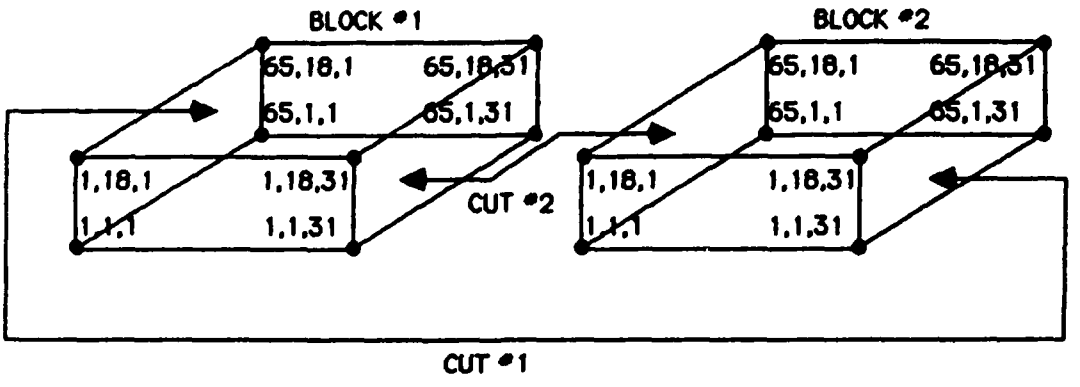


Figure 66. Block to Block Interfaces by Cuts

17. $INPUT ITEM="CUT",BLOCK=1,START=1,1,1,END=65,18,1,IBLOCK=2,ISTART=1,1,31,
IEND=65,18,31$

18. $INPUT ITEM="CUT",BLOCK=1,START=1,1,31,END=65,18,31,IBLOCK=2,ISTART=1,1,1,
IEND=65,18,1$

These lines specify the coordinates for shared boundaries (Interfaces) between blocks 1 and 2.

19. $INPUT ITEM="END"$

This line terminates the inputs to the grid field generation routine and begins the computation of the grid field.

20. $OUTPUT ITEM="ERROR",BLKERR="YES"$

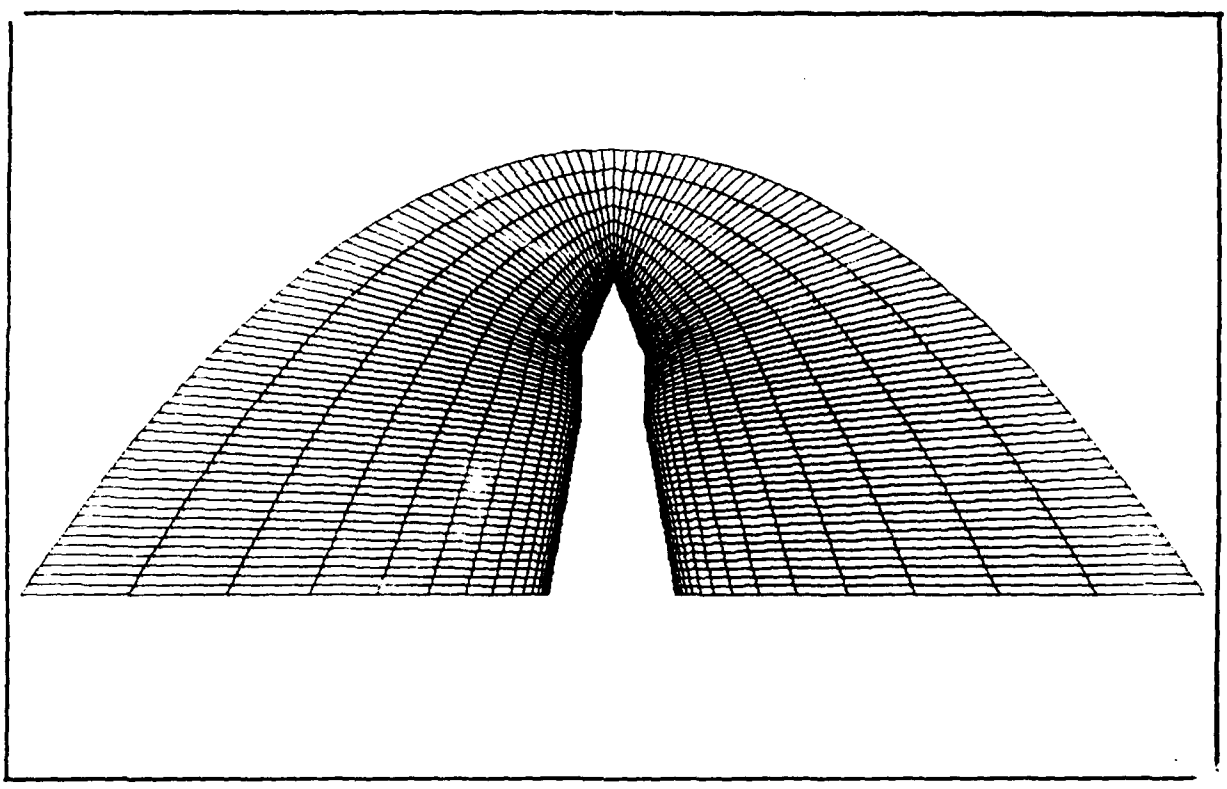
This line requests the grid generation routine to write out the residual of each block (BLKERR="YES") onto the output file.

21. $OUTPUT ITEM="SYSTEM",START=1,1,7,END=65,18,7$

This line writes the coordinates of a plane (START=1,1,7,END=65,18,7) onto the output file.

22. $OUTPUT ITEM="END"$

This line terminates the output.



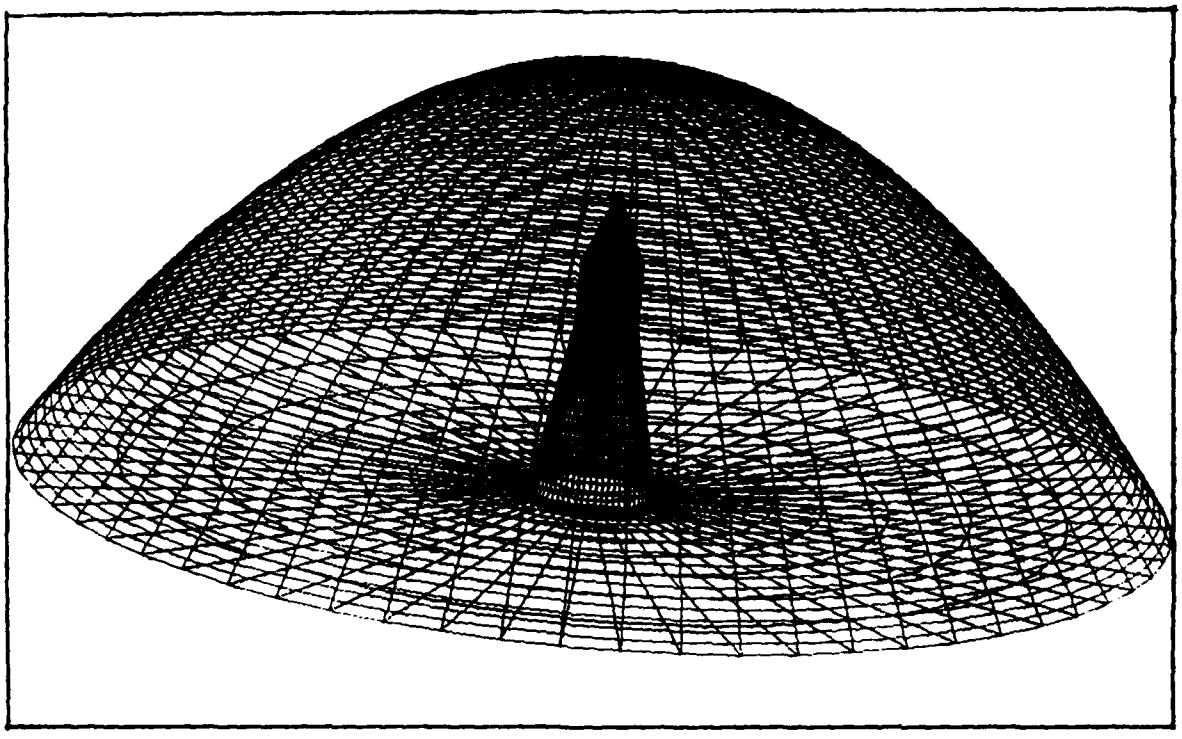


Figure 67. Description of Flow Region Generated by Grid Program

```
$INPUT ITEM="STORE",FILE=75,KSTORE="FILE",ITMAX=1,TOL=1.E-05,
  TRIAD="YES",FORM="E",ACCPAR="OPTIMUM"$
$INPUT ITEM="INITIAL",BLEND="ARC","ARC",CHECK="NO",OUTER="NO",
  CONTYP="RADIUS",ALL="YES"$
*.      BLOCK1
$INPUT ITEM="BLOCK",SIZE=65,18,31$
$INPUT ITEM="FILE",START=1,18,1,END=65,18,31,CLASS="FIX",FILE=14,
  REWIND="YES"$
$INPUT ITEM="FILE",START=1,1,1,END=65,1,31,CLASS="FIX",FILE=14,
  REWIND="NO"$
$INPUT ITEM="FILE",START=65,1,1,END=65,18,31,CLASS="FIX",FILE=13,
  REWIND="YES"$
$INPUT ITEM="FILE",START=1,1,1,END=1,18,31,CLASS="FIX",FILE=12,
  REWIND="YES"$
$INPUT ITEM="INTERP",START=1,1,1,END=65,18,1,BLEND="ARC","ARC","ARC"$
$INPUT ITEM="INTERP",START=1,1,31,END=65,18,31,BLEND="ARC","ARC","ARC"$
*.      BLOCK2
$INPUT ITEM="BLOCK",SIZE=65,18,31$
$INPUT ITEM="FILE",START=1,18,1,END=65,18,31,CLASS="FIX",FILE=15,
  REWIND="YES"$
$INPUT ITEM="FILE",START=1,1,1,END=65,1,31,CLASS="FIX",FILE=15,
  REWIND="NO"$
$INPUT ITEM="FILE",START=65,1,1,END=65,18,31,CLASS="FIX",FILE=13,
  REWIND="NO"$
$INPUT ITEM="FILE",START=1,1,1,END=1,18,31,CLASS="FIX",FILE=12,
  REWIND="NO"$
$INPUT ITEM="INTERP",START=1,1,1,END=65,18,1,BLEND="ARC","ARC","ARC"$
$INPUT ITEM="INTERP",START=1,1,31,END=65,18,31,BLEND="ARC","ARC","ARC"$
*.      CUTS
$INPUT ITEM="CUT",BLOCK=1,START=1,1,1,END=65,18,1,IBLOCK=2,ISTART=1,1,31,
  IEND=65,18,31$
$INPUT ITEM="CUT",BLOCK=1,START=1,1,31,END=65,18,31,IBLOCK=2,ISTART=1,1,1,
  IEND=65,18,1$
$INPUT ITEM="END"$
*.      PLANE THROUGH 3-D SYSTEM
$OUTPUT ITEM="ERROR",BLKERR="YES"$
$OUTPUT ITEM="SYSTEM",START=1,1,7,END=65,18,7$
$OUTPUT ITEM="END"$
```

OGIVE-CYLINDER-OGIVE (4 BLOCKS)

The Program EAGLE Numerical Grid Generation System has the capability to model finned bodies as shown in the following example. This example consists of an ogive-cylinder-ogive body of rotation with a cylindrical sting attachment and four equally spaced fins (Figure 68).

Detailed input for the surface and grid generation systems is presented in the following text.

SURFACE INPUT DATA

```
1.  $INPUT ITEM="SETVAL",NUMBER=1,VALUE=0.03$
2.  $INPUT ITEM="SETVAL",NUMBER=2,VALUE=0.2$
```

Lines 1 and 2 define numerical values used elsewhere in the surface inputs and assigns each value a numerical reference.

```
3.  $INPUT ITEM="SETNUM",SEGMENT=1,POINTS=35$
4.  $INPUT ITEM="SETNUM",SEGMENT=2,POINTS=10$
5.  $INPUT ITEM="SETNUM",SEGMENT=3,POINTS=25$
6.  $INPUT ITEM="SETNUM",SEGMENT=4,POINTS=9$
7.  $INPUT ITEM="SETNUM",SEGMENT=5,POINTS=46$
8.  $INPUT ITEM="SETNUM",SEGMENT=6,POINTS=28$
9.  $INPUT ITEM="SETNUM",SEGMENT=7,POINTS=9$
10. $INPUT ITEM="SETNUM",SEGMENT=8,POINTS=16$
11. $INPUT ITEM="SETNUM",SEGMENT=9,POINTS=9$
12. $INPUT ITEM="SETNUM",SEGMENT=10,POINTS=16$
13. $INPUT ITEM="SETNUM",SEGMENT=11,POINTS=46$
14. $INPUT ITEM="SETNUM",SEGMENT=12,POINTS=9$
15. $INPUT ITEM="SETNUM",SEGMENT=13,POINTS=9$
```

Lines 3 through 15 define the number of grid points on each of the 13 line segments used to define the model surface.

```
16.  $INPUT ITEM="POINT",POINT=1,R=3.3333,1.0,0$
17.  $INPUT ITEM="POINT",POINT=2,R=8.216,1.0,0$
```

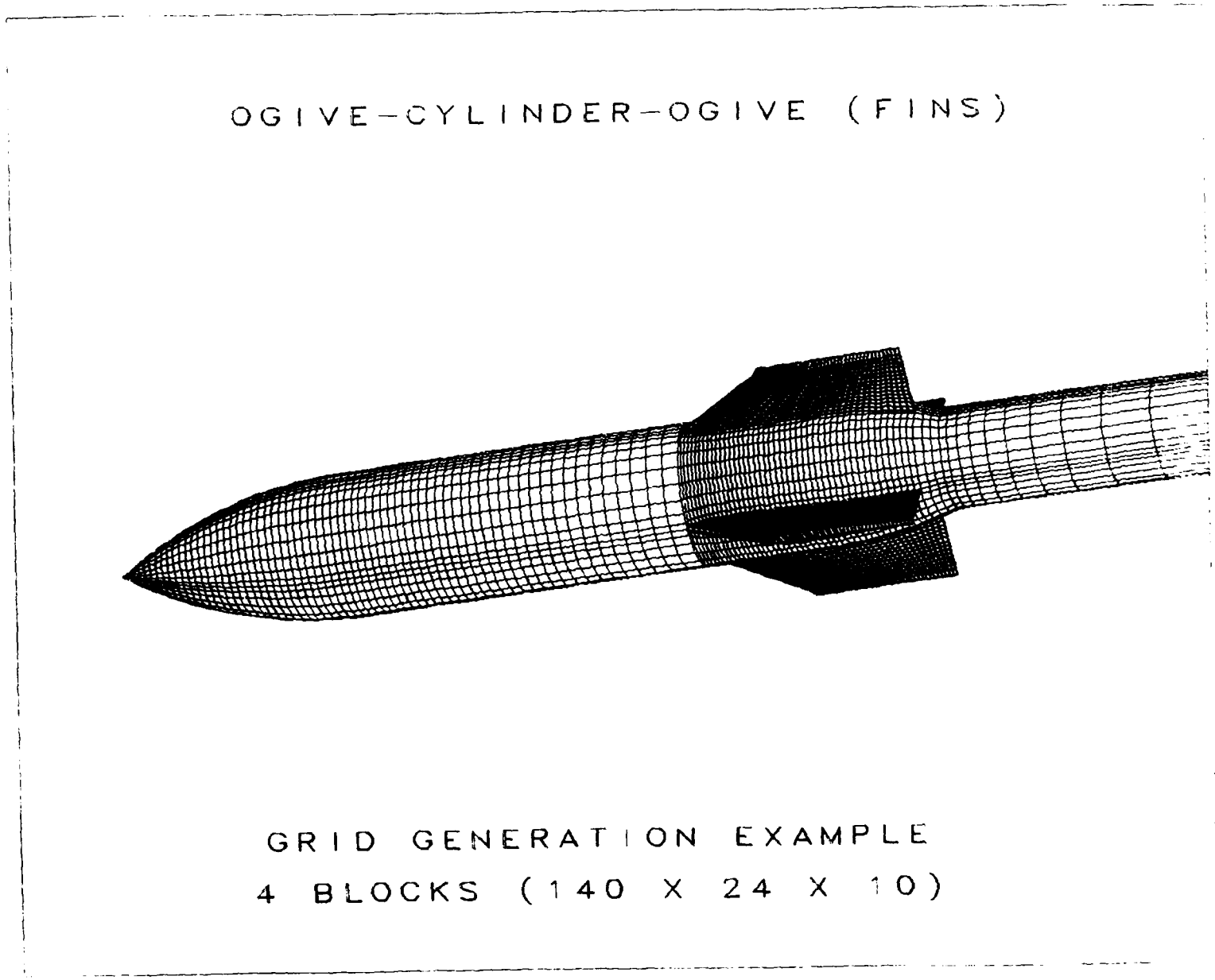


Figure 68. Ogive-Cylinder-Ogive With Fins

```
18.  $INPUT ITEM="POINT",POINT=3,R=10.0,1.0,0$
19.  $INPUT ITEM="POINT",POINT=4,R=11.882,0.7,0$
20.  $INPUT ITEM="POINT",POINT=5,R=72.0,0.7,0$
21.  $INPUT ITEM="POINT",POINT=6,R=40.0,60.0,0$
22.  $INPUT ITEM="POINT",POINT=7,R=3.3333,60.0,0$
23.  $INPUT ITEM="POINT",POINT=8,R=50.0,60.0,0$
24.  $INPUT ITEM="POINT",POINT=9,R=72.0,60.0,0$
25.  $INPUT ITEM="POINT",POINT=10,R=72.0,1.8,0$
26.  $INPUT ITEM="POINT",POINT=11,R=-1.8,0.0,0$
27.  $INPUT ITEM="POINT",POINT=12,R=-56.6667,0.0,0$
28.  $INPUT ITEM="POINT",POINT=13,R=0.0,0.0,0$
29.  $INPUT ITEM="POINT",POINT=14,R=9.51457,1.8,0$
30.  $INPUT ITEM="POINT",POINT=15,R=11.549,1.8,0$
```

Lines 16 through 30 define the spatial location of 15 points used to define the surfaces and assigns each point a numerical reference.

```
31.  $INPUT ITEM="CONICUR",CURPTS=34,TYPE="CIRCLE",RADIUS=6.056,
       ANGLE=56.60428394,90.0$
32.  $INPUT ITEM="TRANS",POINTS=34,ORIGIN=3.3333,-5.056,0,
       COSINES=-1,0,0, 0,1,0, 0,0,-1,COREOUT=1$
```

Lines 31 and 32 create an ogive nose conic section and linearly distributes 34 points on the curve.

The transformation matrix

$$a_{ij} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

rotates this circular arc 180 degrees around the y axis. The change of origin (ORIGIN=3.3333,-5.056,0) translates the curve to its final position. The final curve is stored in temporary memory location (1).

```
33.  $INPUT ITEM="LINE",R1=1,R2=2,POINTS=-1$
```

```
34.  $INPUT ITEM="CURDIST",POINTSI=35,DISTYP="BOTH",RELATIV="NO",
       POINTS=35,SPACE=0.1,0.05,COREOUT=2$
```

Lines 33 and 34 generate a 35-point (POINTS=-1) straight line between point 1 (3.3333,1.0,0) and point 2 (8.216,1.0,0). Grid point spacing is fixed at both ends of the curve. The line is finally stored in temporary memory location (2).

```
35.  $INPUT ITEM="LINE",R1=2,R2=3,POINTS=-2,COREOUT=3$
```

Line 35 generates a 10-point (POINTS = -2) straight line between point 2 (8.216,1.0,0) and point 3 (10.0,1.0,0). The line is then stored in temporary memory location (3).

```
36.  $INPUT ITEM="CONICUR",CURPTS=37,TYPE="CIRCLE",RADIUS=6.056,
       ANGLE=90.0,75.1802273$
37.  $INPUT ITEM="TRANS",POINTS=37,ORIGIN=10.0,-5.056,0,
       COSINES=1,0,0, 0,1,0, 0,0,1,COREOUT=4$
```

Lines 36 and 38 generate a 37-point ogive boattail. The point spacing is linear. The transformation matrix

$$a_{ij} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

does not change the orientation of the curve, but the change of origin (ORIGIN=10.0,-5.056,0) translates the curve to its final position.

The final curve is stored in temporary memory location (4).

```
38.  $INPUT ITEM = "CONICUR",CURPTS=4,TYPE="CIRCLE",RADIUS=6.056,
       ANGLE=75.1802273,71.89458975$
39.  $INPUT ITEM = "TRANS",POINTS=4,ORIGIN=10.0,-5.056,0,
       COSINES=1,0,0, 0,1,0, 0,0,1,COREOUT=5$
```

Lines 38 and 39 generate a ogive-to-sting connection and linearly

distributes 4 points on the line. The curve is then translated (not rotated) and finally stored in temporary memory location (5).

```
40.  $INPUT ITEM="LINE",R1=4,R2=5,POINTS=-3$
41.  $INPUT ITEM="CURDIST",POINTSI=25,DISTYP="TANH",RELATIV="NO",
       POINTS=25,SPACE=0.10,COREOUT=6$
```

Lines 40 and 41 generate a 25-point (POINTS=-3) sting from point 4 (11.882,0.7,0) to point 5 (72.0,0.7,0) with a hyperbolic tangent distribution concentrating points near the ogive-cylinder-ogive body.

The line segment is stored in temporary memory location (6).

```
42.  $INPUT ITEM="CURRENT",COREIN=1,POINTS=34$
43.  $INPUT ITEM="INSERT",COREIN=2,POINTSI=35,START=34,POINTS=68$
44.  $INPUT ITEM="INSERT",COREIN=3,POINTSI=10,START=68,POINTS=77$
45.  $INPUT ITEM="INSERT",COREIN=4,POINTSI=37,START=77,POINTS=113$
46.  $INPUT ITEM="INSERT",COREIN=5,POINTSI=4,START=113,POINTS=116$
47.  $INPUT ITEM="INSERT",COREIN=6,POINTSI=25,START=116,POINTS=140,
       COREOUT=10$
```

Lines 42 through 47 combine the six previously generated curves into one curve which defines the two-dimensional outline of the ogive-cylinder-ogive body with sting (Figure 69). The combined curve is stored in temporary memory location (10).

```
48.  $INPUT ITEM="CURRENT",POINTS=250,
                    VALUES = 8.216000, 1.000000, 0.000000,
                             8.229360, 1.000000, 0.024381,
                                 :         :         :
                                 :         :         :
                                 :         :         :
                            11.529260, 0.808795, 0.005479,
                            11.542620, 0.805339, 0.004030$
49.  $INPUT ITEM="CURDIST",POINTSI=250,POINTS=46,
       DISTYP="TANH",SPACE=.05,RELATIV="NO",COREOUT=9$
```

Lines 48 and 49 generate a curve which defines one-half of the symmetric fin section at the root. A list of 250 points defining the curve is read in and

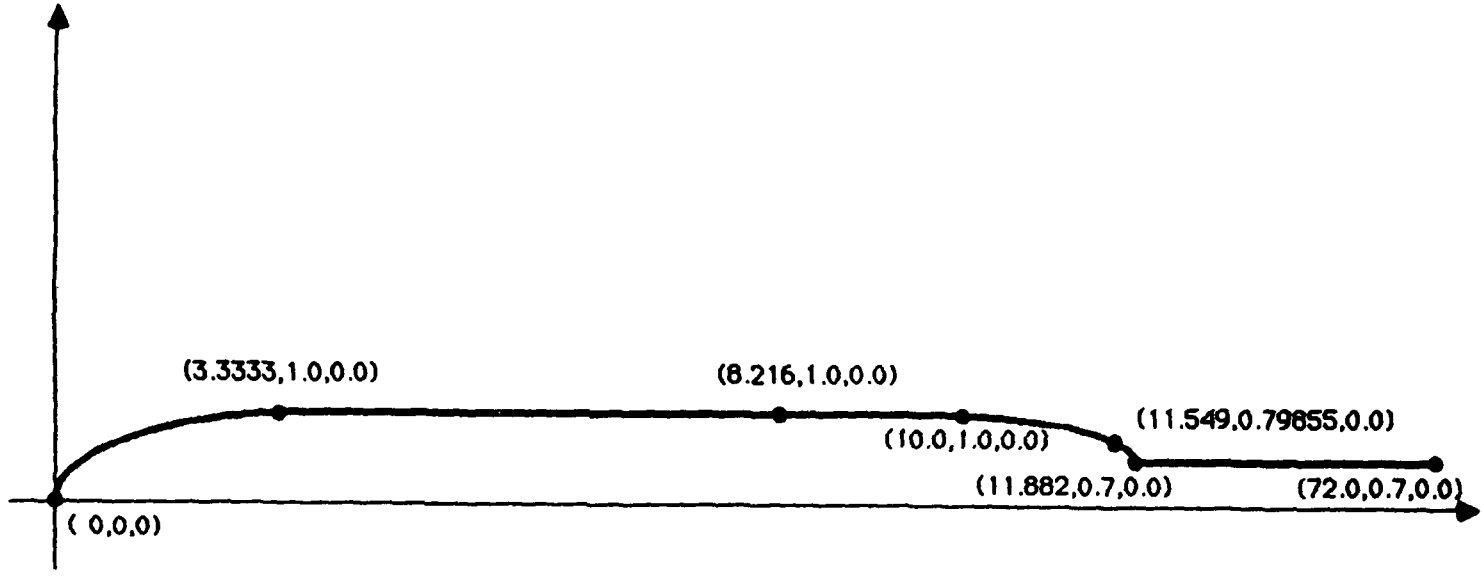
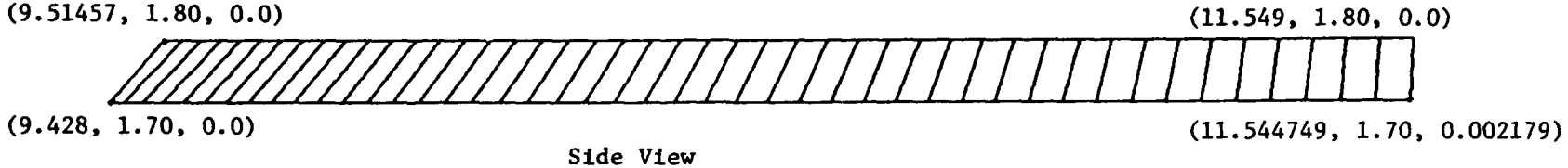


Figure 69. Ogive-Cylinder-Ogive Boundary With Sting

Top View



Side View

Figure 70. 46 x 2 Point Fin Tip Surface

then 46 points are distributed over the curve using a hyperbolic tangent distribution to concentrate points near the leading edge. This final curve is then stored in temporary memory location (9).

```
50.  $INPUT ITEM="SCALE",POINTS=46,SCALE=1,1,-1,COREOUT=8$
```

Line 50 generates a mirror image reflection of the fin root section across the z axis (Figure 70), and saves it in temporary memory location (8).

```
51.  $INPUT ITEM="CURRENT",COREIN=10,POINTS=140$
52.  $INPUT ITEM="INSERT",COREIN=8,POINTSI=46,START=68,POINTS=140,
       COREOUT=7$
```

Lines 51 and 52 call in the 140-point outline of the ogive-cylinder-ogive body with sting and insert the 46-point fin root section starting at point 68. The result is stored in temporary memory location (7).

```
53.  $INPUT ITEM="CURRENT",COREIN=10,POINTS=140$
54.  $INPUT ITEM="INSERT",COREIN=9,POINTSI=46,START=68,POINTS=140,
       COREOUT=6$
```

Lines 53 and 54 are analogous to 51 and 52 but insert the mirror image fin root section and store the result in temporary memory location (6).

```
55.  $INPUT ITEM="CURRENT",POINTS=250,
                    VALUES = 9.428000, 1.700000, 0.000000,
                             9.436501, 1.700000, 0.015514,
                                 :         :         :
                                 :         :         :
                                 :         :         :
                            11.536248, 1.700000, 0.002970,
                            11.544749, 1.700000, 0.002179$

56.  $INPUT ITEM = "CURDIST",POINTSI=250,POINTS=46,
       DISTYP="TANH",SPACE=.0318,RELATIV="NO",COREOUT=3$
```

Lines 55 and 56 read in the coordinates of the 250-point curve defining one-

half of the symmetric airfoil section at the fin tip. Forty-six points are distributed on the curve using a hyperbolic tangent distribution to concentrate points near the leading edge. The final result is stored in temporary memory location (3).

```
57.  $INPUT ITEM = "SCALE",POINTS=46,SCALE=1,1,-1,COREOUT=5$
```

Line 57 reflects a mirror image of the fin-tip, airfoil section for use on the other side of the Block. This curve is stored in temporary memory location (5).

```
58.  $INPUT ITEM="LINE",R1=14,R2=15,POINTS=-11,
       DISTYP="TANH",SPACE=-1,RELATIV="NO",COREOUT=2$
```

Line 58 generates a 46-point (POINTS = -11) line from point 14 (9.51457,1.8,0) to point 15 (11.549,1.8,0) to be used to close off the fin tip. Points are concentrated near the leading edge of the line using a hyperbolic tangent distribution. The completed line is stored in temporary memory location (2).

```
59.  $INPUT ITEM="BOUNCUR",COREIN=3,POINTS=46$
60.  $INPUT ITEM="BOUNCUR",COREIN=2,POINTS=46$
61.  $INPUT ITEM="BLEND",POINTS=46,CURVES=2,COREOUT=1$
```

Lines 59 through 61 create a 46 point by 2 point surface by blending the fin-tip section and the line that was created in input line 58 (Figure). The surface is stored in temporary memory location (1).

```
62.  $INPUT ITEM="BOUNCUR",COREIN=9,POINTS=46$
63.  $INPUT ITEM="BOUNCUR",COREIN=3,POINTS=46$
64.  $INPUT ITEM="BLEND",POINTS=46,CURVES=15$
```

Lines 62 through 64 generate one-half of the fin surface by blending the fin tip section and the fin root section with 15 intermediate airfoil sections (Figure 71).

```
65.  $INPUT ITEM="INSERT",COREIN=1,POINTSI=46,2,START=1,15,POINTS=46,16,
       COREOUT=4$
```

Line 65 inserts the 46 x 2 point fin tip surface (stored in temporary memory location 1) into the 46 x 15 point fin surface yielding a complete 46 x 16 point fin surface which is stored in temporary memory location (4).

```
66.  $INPUT ITEM="BOUNCUR",COREIN=5,POINTS=46$
67.  $INPUT ITEM="BOUNCUR",COREIN=2,POINTS=46$
68.  $INPUT ITEM="BLEND",POINTS=46,CURVES=2,COREOUT=1$
69.  $INPUT ITEM="BOUNCUR",COREIN=8,POINTS=46$
70.  $INPUT ITEM="BOUNCUR",COREIN=5,POINTS=46$
71.  $INPUT ITEM="BLEND",POINTS=46,CURVES=15$
72.  $INPUT ITEM="INSERT",COREIN=1,POINTSI=46,2,START=1,15,POINTS=46,16,
       COREOUT=5$
```

Lines 66 through 72 accomplish the same tasks as lines 59 through 65 but for a mirror image fin surface to complete the symmetric-airfoil fin. The airfoil symmetry allows the axisymmetric body to be broken into four blocks with the block boundaries being the fin symmetry plane. The resulting surface is stored in temporary memory location (5).

```
73.  $INPUT ITEM="BOUNCUR",COREIN=6,POINTS=140$
74.  $INPUT ITEM="BOUNCUR",COREIN=7,POINTS=140$
75.  $INPUT ITEM="ROTATE",CURPTS=140,ANGPTS=10,ANGLE=-45,45,
       DISTANG="BOTH",SPACANG=0.05,0.05,AXCOS=1,0,0,NORCOS=0,1,0,
       FILEOUT=1$
```

Lines 73 and 74 designate the 140-point curves defining the outline of the ogive-cylinder-ogive with sting (with fin root section and mirror image fin root section) as bounding curves for a surface generation. Line 75 creates a surface of revolution by interpolating the bounding curves and forms the missile body surface from -45 to 45 degrees(Figure 72). The 140 x 10 point surface is stored in output File (1).

```
76.  $INPUT ITEM="BOUNCUR",COREIN=6,POINTS=140$
```

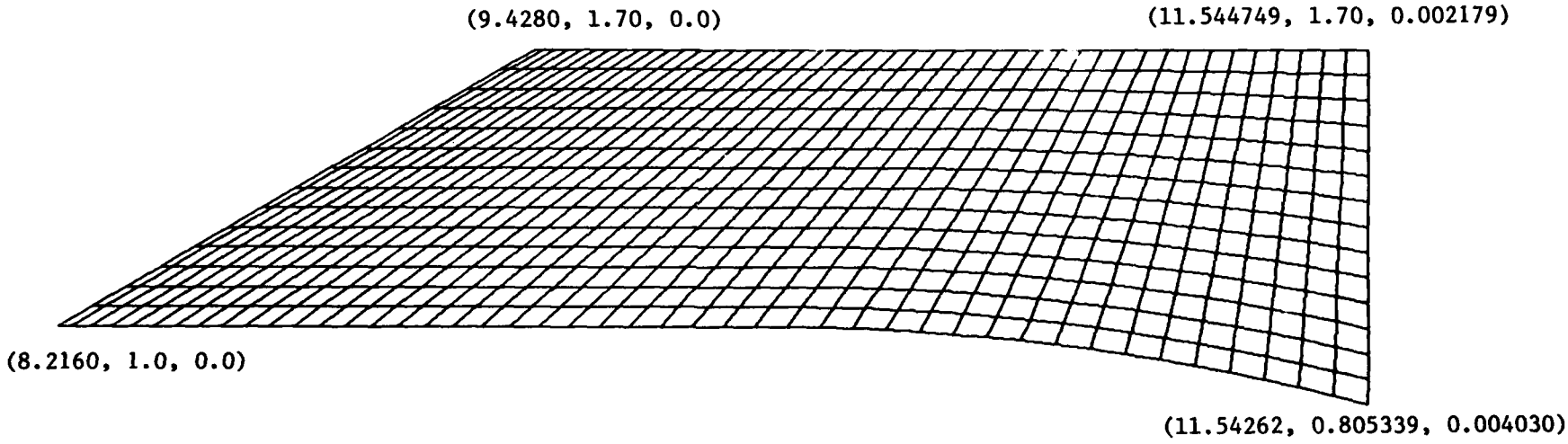


Figure 71. Fin Surface

```
77.  $INPUT ITEM="BOUNCUR",COREIN=7,POINTS=140$
78.  $INPUT ITEM="ROTATE",CURPTS=140,ANGPTS=10,ANGLE=45,135,
       DISTANG="BOTH",SPACANG=0.05,0.05,AXCOS=1,0,0,NORCOS=0,1,0,
       FILEOUT=2$
```

Lines 76 through 78 generate the missile body surface from 45 to 135 degrees. The surface is stored in output File (2).

```
79.  $INPUT ITEM="BOUNCUR",COREIN=6,POINTS=140$
80.  $INPUT ITEM="BOUNCUR",COREIN=7,POINTS=140$
81.  $INPUT ITEM="ROTATE",CURPTS=140,ANGPTS=10,ANGLE=135,225,
       DISTANG="BOTH",SPACANG=0.05,0.05,AXCOS=1,0,0,NORCOS=0,1,0,
       FILEOUT=3$
```

Lines 79 through 81 generate the missile body surface from 135 to 225 degrees. The surface is then stored in output File (3).

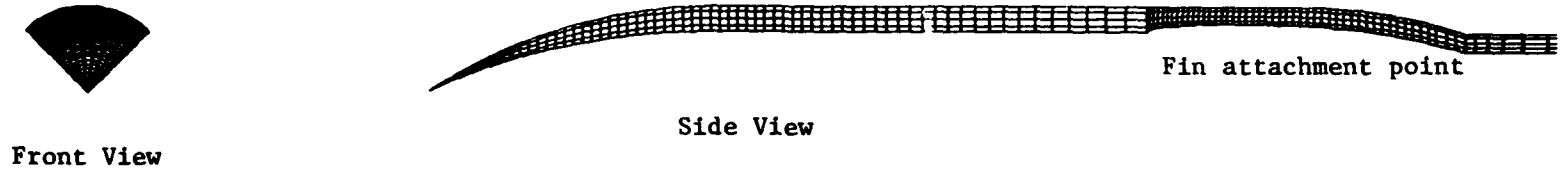
```
82.  $INPUT ITEM="BOUNCUR",COREIN=6,POINTS=140$
83.  $INPUT ITEM="BOUNCUR",COREIN=7,POINTS=140$
84.  $INPUT ITEM="ROTATE",CURPTS=140,ANGPTS=10,ANGLE=225,315,
       DISTANG="BOTH",SPACANG=0.05,0.05,AXCOS=1,0,0,NORCOS=0,1,0,
       FILEOUT=4$
```

Lines 82 through 84 generate the missile body surface from 225 to 315 degrees. The surface is then stored in output File (4).

```
85.  $INPUT ITEM="TRANS",COREIN=5,POINTS=46,16,
       COSINES=1,0,0, 0,.707,.707, 0,-.707,.707, FILEOUT=5$
86.  $INPUT ITEM="TRANS",COREIN=4,POINTS=46,16,
       COSINES=1,0,0, 0,.707,.707, 0,-.707,.707, FILEOUT=6$
```

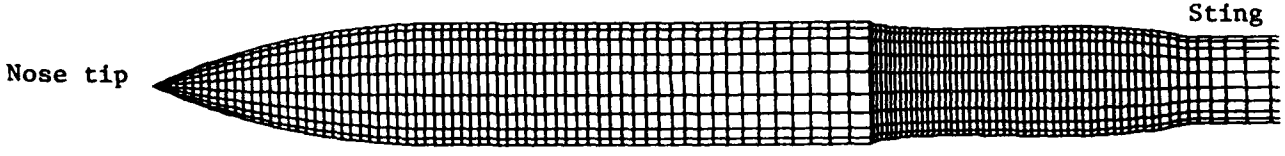
Lines 85 and 86 place the 46 x 16 fin surfaces at the +45 degree position for use as the fin boundaries between Blocks 1 and 2. The new fin surfaces are stored in output Files 5 and 6.

```
87.  $INPUT ITEM="TRANS",COREIN=5,POINTS=46,16,
       COSINES=1,0,0, 0,.707,-.707,0,.707,.707,FILEOUT=7$
```



Front View

Side View



Top View

Figure 72. Missile Body Surface

```
88.  $INPUT ITEM="TRANS",COREIN=4,POINTS=46,16,
       COSINES=1,0,0, 0,.707,-.707, 0,.707,.707,FILEOUT=8$
```

Lines 87 and 88 place the 46 x 16 fin surfaces at the -45 degree position for use as the fin boundaries between Blocks 1 and 4. The new fin surfaces are stored in output Files 7 and 8.

```
89.  $INPUT ITEM="TRANS",COREIN=5,POINTS=46,16,
       COSINES=1,0,0, 0,-.707,-.707,0,.707,-.707,FILEOUT=9$
90.  $INPUT ITEM="TRANS",COREIN=4,POINTS=46,16,
       COSINES=1,0,0, 0,-.707,-.707, 0,.707,-.707,FILEOUT=10$
```

Lines 89 and 90 place the 46 x 16 fin surfaces at the 225 degree position for use as the fin boundaries between Blocks 3 and 4. The new fin surfaces are stored in output Files 9 and 10.

```
91.  $INPUT ITEM="TRANS",COREIN=5,POINTS=46,16,
       COSINES=1,0,0, 0,-.707,.707, 0,-.707,-.707,FILEOUT=33$
92.  $INPUT ITEM="TRANS",COREIN=4,POINTS=46,16,
       COSINES=1,0,0, 0,-.707,.707, 0,-.707,-.707,FILEOUT=35$
```

Lines 85 and 86 place the 46 x 16 fin surfaces at the 135 degree position for use as the fin boundaries between Blocks 2 and 3. The new fin surfaces are stored in output Files 33 and 35.

```
93.  $INPUT ITEM="LINE",R1=14,R2=6,POINTS=-12$
94.  $INPUT ITEM="CURDIST",POINTSI=9,DISTYP="TANH",RELATIV="NO",
       SPACE=0.1,POINTS=9,COREOUT=1$
```

Lines 93 and 94 generate a line from the leading edge of the fin tip to the outer boundary with points concentrated near the fin tip. The line runs from point 14 (9.51457,1.8,0) to point 6 (40.0,60.0,0) and has 9 points (POINTS = -12). The purpose of the line is to control the grid lines coming off the fin tip. The line is then stored in temporary memory location (1).

```
95.  $INPUT ITEM="TRANS",COREIN=1,POINTS=9,
       COSINES=1,0,0, 0,.707,.707, 0,-.707,.707,FILEOUT=34$
```

```
96.  $INPUT ITEM="TRANS",COREIN=1,POINTS=9,
       COSINES=1,0,0, 0,.707,-.707, 0,.707,.707,FILEOUT=14$
97.  $INPUT ITEM="TRANS",COREIN=1,POINTS=9,
       COSINES=1,0,0, 0,-.707,-.707, 0,.707,-.707,FILEOUT=15$
98.  $INPUT ITEM="TRANS",COREIN=1,POINTS=9,
       COSINES=1,0,0, 0,-.707,.707, 0,-.707,-.707,FILEOUT=16$
```

Lines 95 through 98 place the leading edge controlling line at -45, +45, 135, and 225 degree fin positions. The lines are stored in output files 34, 14, 15, and 16 respectively.

```
99.  $INPUT ITEM="LINE",R1=15,R2=8,POINTS=-13$
100. $INPUT ITEM="CURDIST",POINTSI=9,DISTYP="TANH",RELATIV="NO",
       SPACE=0.1,POINTS=9,COREOUT=1$
101. $INPUT ITEM="TRANS",COREIN=1,POINTS=9,
       COSINES=1,0,0, 0,.707,.707, 0,-.707,.707,FILEOUT=17$
102. $INPUT ITEM="TRANS",COREIN=1,POINTS=9,
       COSINES=1,0,0, 0,.707,-.707, 0,.707,.707,FILEOUT=18$
103. $INPUT ITEM="TRANS",COREIN=1,POINTS=9,
       COSINES=1,0,0, 0,-.707,-.707, 0,.707,-.707,FILEOUT=19$
104. $INPUT ITEM="TRANS",COREIN=1,POINTS=9,
       COSINES=1,0,0, 0,-.707,.707, 0,-.707,-.707,FILEOUT=20$
```

Lines 99 through 104 are analogous to lines 93 through 98, but generate controlling grid lines from the trailing edge of each fin to the outer boundary. The lines are stored in output files 17 through 20 respectively.

```
105. $INPUT ITEM="CONICUR",CURPTS=60,TYPE="CIRCLE",
       RADIUS=60.0,ANGLE=90.0,0.0$
106. $INPUT ITEM="TRANS",POINTS=60,ORIGIN=3.3333,0,0,
       COSINES=-1,0,0, 0,1,0, 0,0,-1,COREOUT=1$
```

Lines 105 and 106 generate a circular arc which will be the upstream edge of the outer boundary. The arc is translated (ORIGIN=3.3333,0,0) and rotated around the y axis using the transformation matrix

$$a_{ij} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

Sixty points are linearly distributed on the arc. The resulting curve is stored in temporary memory location (1).

```
107. $INPUT ITEM="LINE",R1=6,R2=7,POINTS=-4$
```

Line 107 generates a line which continues the outer boundary. The line is from point 6 (40.0,60.0,0) to point 7 (3.3333,60.0,0) and has 9 points (POINTS= -4) linearly distributed on it.

```
108. $INPUT ITEM="INSERT",COREIN=1,POINTSI=60,START=9,POINTS=68$
```

Line 108 combines the circular arc and line just generated for the outer boundary and linearly distributes 68 points on the new curve.

```
109. $INPUT ITEM="CURDIST",POINTSI=68,DISTYP="TANH",RELATIV="NO",
       POINTS=68,SPACE=0.2$
```

Line 109 changes the point distribution on the outer boundary from a linear to a hyperbolic tangent distribution with points concentrated near the nose.

```
110. $INPUT ITEM="SWITCH",POINTS=68,REORDER="REVERSE1",COREOUT=1$
```

Line 110 reorders the numbering of points on the outside boundary.

```
111. $INPUT ITEM="LINE",R1=6,R2=8,POINTS=-5,
       DISTYP="BOTH",RELATIV="NO",SPACE=-2,-2,COREOUT=2$
```

Line 111 draws a line from point 6 (40.0,60.0,0) and point 8 (50.0,60.0,0) and distributes 46 points (POINTS = -5) on the line. The point spacing at each end of the line is specified. The result is stored in temporary memory location (2).

```
112. $INPUT ITEM="LINE",R1=8,R2=9,POINTS=-6$
```

Line 112 draws a line from point 8 (50.0,60.0,0) and point 9 (72.0,60.0,0) and linearly distributes 28 points (POINTS = -6) on it.

```
113. $INPUT ITEM="CURDIST",POINTSI=28,DISTYP="TANH",RELATIV="NO",
       SPACE=0.2,POINTS=28,COREOUT=11$
```

Line 113 distributes 28 points over the last segment of the outer boundary. The points are distributed using a hyperbolic tangent function. The final curve is stored in temporary memory location (11).

```
114. $INPUT ITEM="CURRENT",COREIN=1,POINTS=68$
115. $INPUT ITEM="INSERT",COREIN=2,POINTSI=46,START=68,POINTS=113$
116. $INPUT ITEM="INSERT",COREIN=11,POINTSI=28,START=113,POINTS=140,
       COREOUT=11$
```

Lines 114 through 116 combine all segments of the outer boundary into one curve with 140 points (Figure 73).

```
117. $INPUT ITEM="BOUNCUR",POINTS=140$
118. $INPUT ITEM="ROTATE",CURPTS=140,ANGPTS=10,ANGLE=-45,45,
       DISTANG="BOTH",SPACANG=0.05,0.05,AXCOS=1,0,0,NORCOS=0,1,0,FILEOUT=21$
```

Lines 117 and 118 rotate the 140-point outer boundary between -45 and +45 degrees to create the outer boundary for Block 1 (Figure 74). The surface is then stored in File 21 (FILEOUT=21).

```
119. $INPUT ITEM="LINE",R1=10,R2=9,POINTS=-7$
120. $INPUT ITEM="CURDIST",POINTSI=9,DISTYP="TANH",RELATIV="NO",
       POINTS=9,SPACE=0.1,COREOUT=12$
121. $INPUT ITEM="LINE",R1=5,R2=10,POINTS=-8$
122. $INPUT ITEM="INSERT",COREIN=12,POINTSI=9,START=16,POINTS=24,
       COREOUT=14$
```

Lines 119 through 122 create a 24-point back boundary with grid points

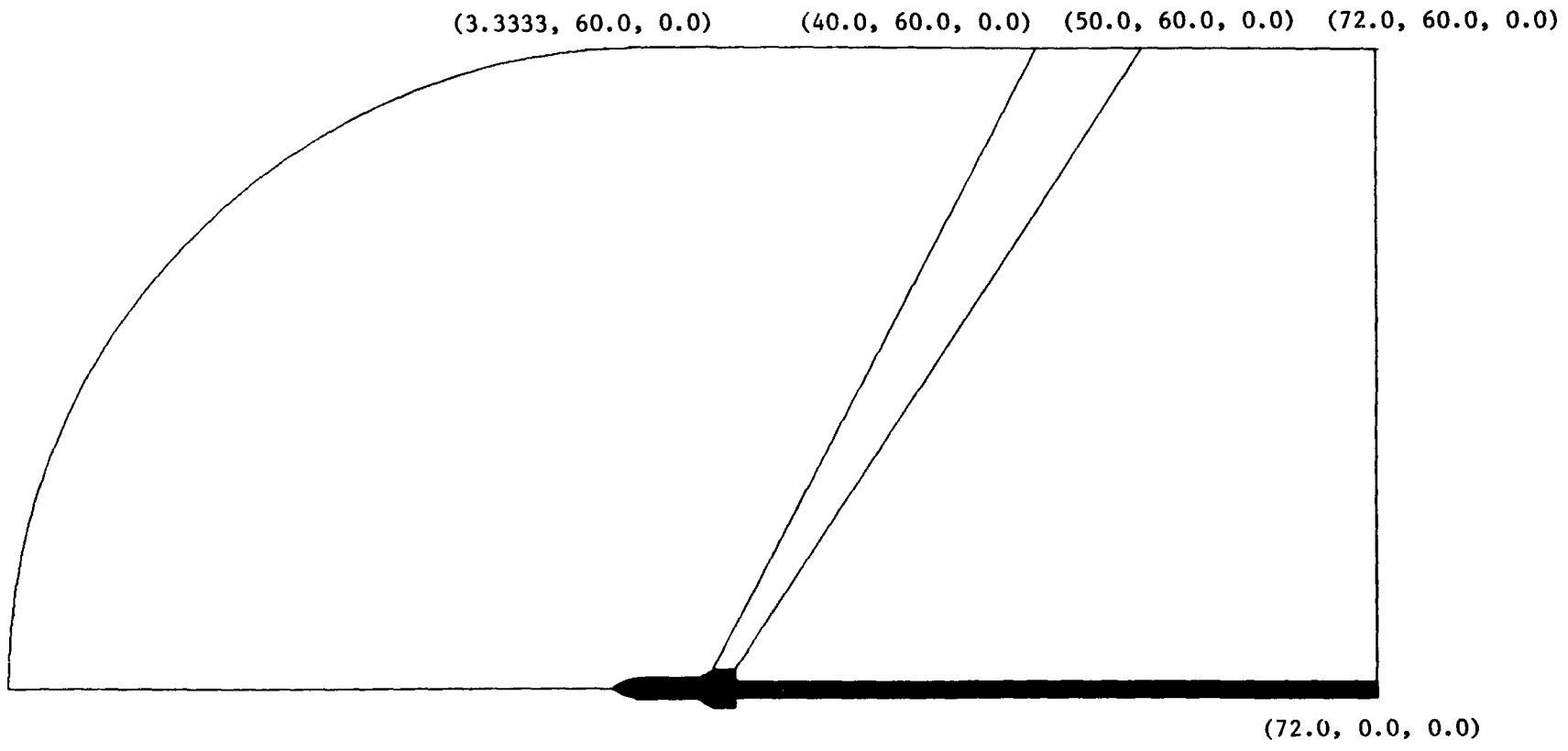


Figure 73. Outer Boundary Outline

concentrated in the fin region. Line 119 generates a line from point 10 (72.0,1.8,0) to point 9 (72.0,60.0,0) with 9 points (POINTS=-7). Line 120 redistributes 9 points on this line using a hyperbolic tangent function to concentrate points near the fin. Line 121 generates a line from point 5 (72.0,0.7,0) to point 10 (72.0,1.8,0) and linearly distributes 16 points (POINTS=-8) on the line. Line 122 combines the current line with the line in temporary memory location 12.

```
123. $INPUT ITEM="BOUNCUR",POINTS=24$
124. $INPUT ITEM="ROTATE",CURPTS=24,ANGPTS=10,ANGLE=-45,45,
       DISTANG="BOTH",SPACANG=0.05,0.05,AXCOS=1,0,0,NORCOS=0,1,0,
       FILEOUT=22$
```

Lines 123 and 124 rotate the 24-point back boundary line with 10 angular points between -45 and +45 degrees to create the Block 1 back boundary surface (Figure 75). The surface is stored in File 22 (FILEOUT=22).

```
125. $INPUT ITEM="LINE",R1=11,R2=12,POINTS=-9$
126. $INPUT ITEM="CURDIST",POINTSI=9,DISTYP="TANH",RELATIV="NO",
       POINTS=9,SPACE=0.1,COREOUT=13$
127. $INPUT ITEM="LINE",R1=13,R2=11,POINTS=-10$
128. $INPUT ITEM="INSERT",COREIN=13,POINTSI=9,START=16,POINTS=24,
       COREOUT=15$
```

Lines 125 through 128 generate a 24-point front stagnation line boundary with point concentrated in the fin region. The line is stored in temporary memory location (15). Line 125 creates a line from point 11 (-1.8,0.0,0) to point 12 (-56.6667,0.0,0) with a linear distribution of 9 points (POINTS=-9). Line 126 distributes 9 points using a hyperbolic tangent distribution to concentrate points in the fin region. Line 127 creates a line from point 13 (0.0,0.0,0) to point 11 (-1.8,0.0,0) with a linear distribution of 16 points (POINTS=-10). Line 128 inserts the line created in line 125 into the current line and stores the result in temporary memory location (15).

```
129. $INPUT ITEM="BOUNCUR",POINTS=24$
```

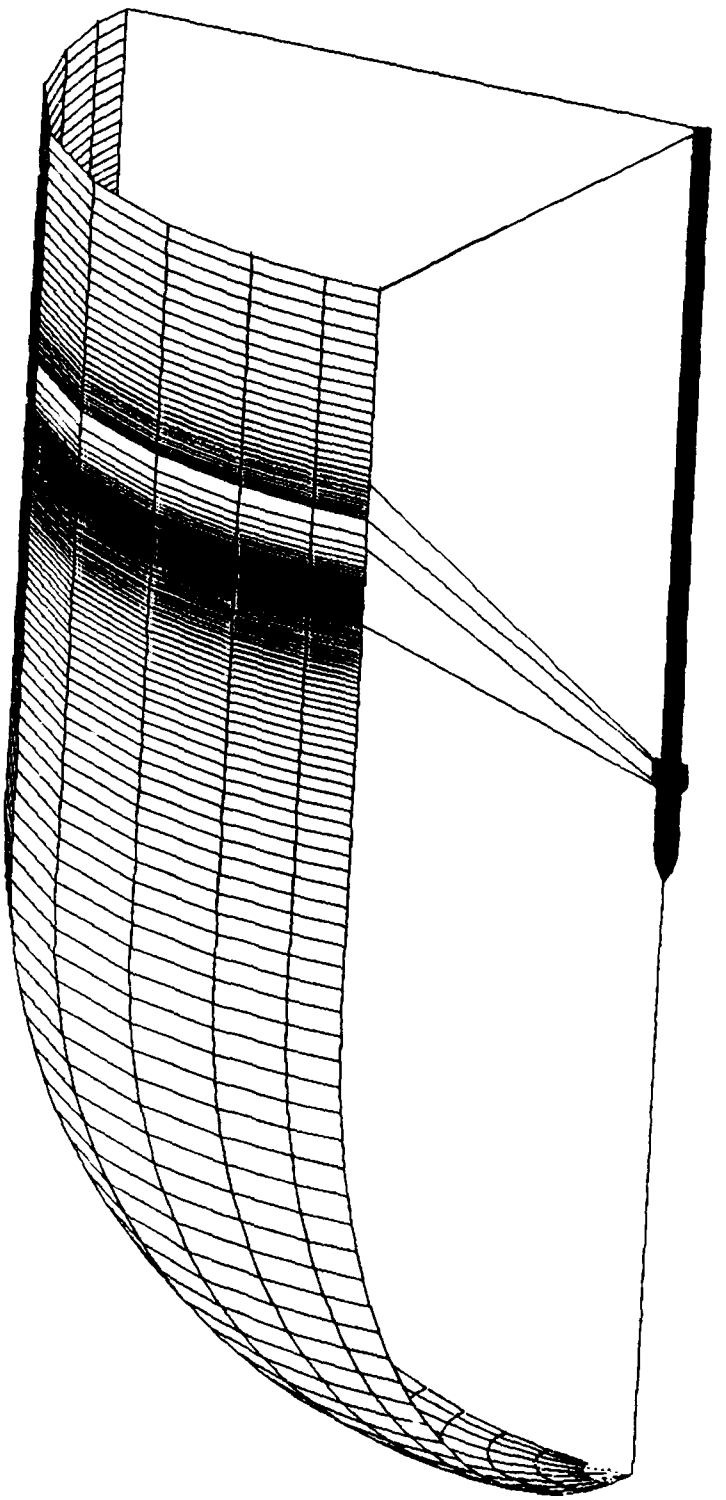


Figure 74. Outer Boundary Surface

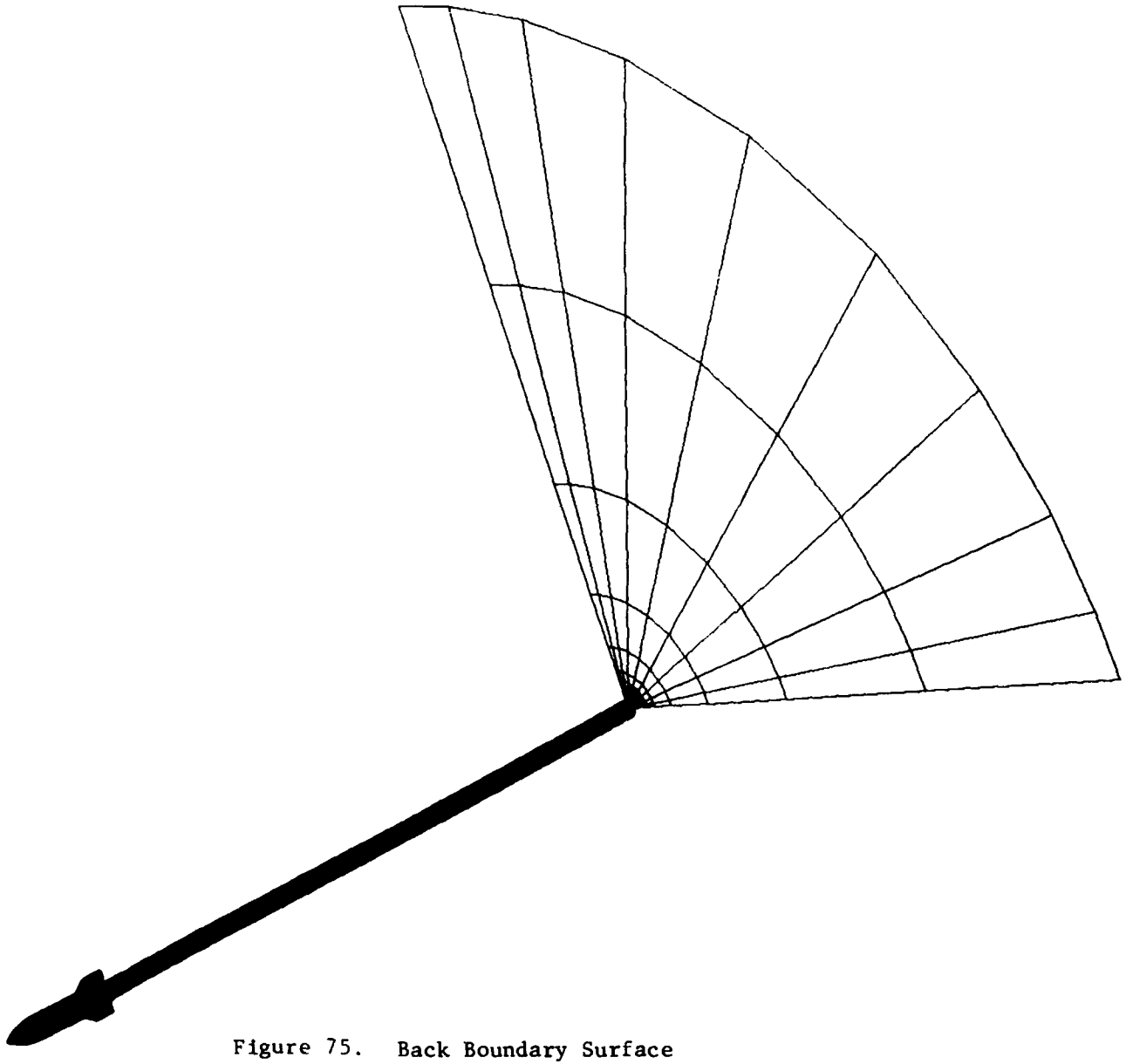


Figure 75. Back Boundary Surface

```
130. $INPUT ITEM="ROTATE",CURPTS=24,ANGPTS=10,ANGLE=-45,45,
       DISTANG="BOTH",SPACANG=0.05,0.05,AXCOS=-1,0,0,NORCOS=0,1,0,
       RMAX=100,100,100,FILEOUT=23$
```

Lines 129 through 130 rotate the 24-point front stagnation line from -45 to +45 degrees with 10 angular points to create the front boundary for Block 1. The result is stored in File 23.

```
131. $INPUT ITEM="BOUNCUR",COREIN=11,POINTS=140$
132. $INPUT ITEM="ROTATE",CURPTS=140,ANGPTS=10,ANGLE=45,135,
       DISTANG="BOTH",SPACANG=0.05,0.05,AXCOS=1,0,0,NORCOS=0,1,0,
       FILEOUT=24$
133. $INPUT ITEM="BOUNCUR",COREIN=14,POINTS=24$
134. $INPUT ITEM="ROTATE",CURPTS=24,ANGPTS=10,ANGLE=45,135,
       DISTANG="BOTH",SPACANG=0.05,0.05,AXCOS=1,0,0,NORCOS=0,1,0,
       FILEOUT=25$
135. $INPUT ITEM="BOUNCUR",COREIN=15,POINTS=24$
136. $INPUT ITEM="ROTATE",CURPTS=24,ANGPTS=10,ANGLE=45,135,
       DISTANG="BOTH",SPACANG=0.05,0.05,AXCOS=-1,0,0,NORCOS=0,1,0,
       FILEOUT=26$
```

Lines 131 through 136 create the outer boundary, the front stagnation line boundary, and the back boundary for Block 2.

```
137. $INPUT ITEM="BOUNCUR",COREIN=11,POINTS=140$
138. $INPUT ITEM="ROTATE",CURPTS=140,ANGPTS=10,ANGLE=135,225,
       DISTANG="BOTH",SPACANG=0.05,0.05,AXCOS=1,0,0,NORCOS=0,1,0,
       FILEOUT=27$
139. $INPUT ITEM="BOUNCUR",COREIN=14,POINTS=24$
140. $INPUT ITEM="ROTATE",CURPTS=24,ANGPTS=10,ANGLE=135,225,
       DISTANG="BOTH",SPACANG=0.05,0.05,AXCOS=1,0,0,NORCOS=0,1,0,
       FILEOUT=28$
141. $INPUT ITEM="BOUNCUR",COREIN=15,POINTS=24$
142. $INPUT ITEM="ROTATE",CURPTS=24,ANGPTS=10,ANGLE=135,225,
       DISTANG="BOTH",SPACANG=0.05,0.05,AXCOS=-1,0,0,NORCOS=0,1,0,
       FILEOUT=29$
```

Lines 137 through 142 create the outer boundary, the front stagnation line boundary, and the back boundary for Block 3.

```
143. $INPUT ITEM="BOUNCUR",COREIN=11,POINTS=140$
144. $INPUT ITEM="ROTATE",CURPTS=140,ANGPTS=10,ANGLE=225,315,
       DISTANG="BOTH",SPACANG=0.05,0.05,AXCOS=1,0,0,NORCOS=0,1,0,
       FILEOUT=30$
145. $INPUT ITEM="BOUNCUR",COREIN=14,POINTS=24$
146. $INPUT ITEM="ROTATE",CURPTS=24,ANGPTS=10,ANGLE=225,315,
       DISTANG="BOTH",SPACANG=0.05,0.05,AXCOS=1,0,0,NORCOS=0,1,0,
       FILEOUT=31$
147. $INPUT ITEM="BOUNCUR",COREIN=15,POINTS=24$
148. $INPUT ITEM="ROTATE",CURPTS=24,ANGPTS=10,ANGLE=225,315,
       DISTANG="BOTH",SPACANG=0.05,0.05,AXCOS=-1,0,0,NORCOS=0,1,0,
       FILEOUT=32$
```

Lines 143 through 148 create the outer boundary, the front stagnation line boundary, and the back boundary for Block 4.

```
149. $INPUT ITEM="COMBINE", FILEIN=1,21,22,23,14,18,34,17,8,5,
       FILEOUT=40$
150. $INPUT ITEM="COMBINE",FILEIN=2,24,25,26,34,17,16,20,6,33,
       FILEOUT=50$
151. $INPUT ITEM="COMBINE",FILEIN=3,27,28,29,16,20,15,19,35,9,FILEOUT=60$
152. $INPUT ITEM="COMBINE",FILEIN=4,30,31,32,15,19,14,18,10,7,
       FILEOUT=70$
```

Lines 149 through 152 combine all the output files for a given block (Blocks 1 through 4 respectively) into one output file (Files 40 through 70 respectively).

```
153. $INPUT ITEM="END"$
```

Line 153 terminates the input to the surface generation code.

```
SINPUT ITEM = "SETVAL", NUMBER = 1, VALUE = 0.03 S
SINPUT ITEM = "SETVAL", NUMBER = 2, VALUE = 0.2 S
SINPUT ITEM = "SETNUM", SEGMENT = 1, POINTS = 35$
SINPUT ITEM = "SETNUM", SEGMENT = 2, POINTS = 10$
SINPUT ITEM = "SETNUM", SEGMENT = 3, POINTS = 25$
SINPUT ITEM = "SETNUM", SEGMENT = 4, POINTS = 9$
SINPUT ITEM = "SETNUM", SEGMENT = 5, POINTS = 46$
SINPUT ITEM = "SETNUM", SEGMENT = 6, POINTS = 28$
SINPUT ITEM = "SETNUM", SEGMENT = 7, POINTS = 9$
SINPUT ITEM = "SETNUM", SEGMENT = 8, POINTS = 16$
SINPUT ITEM = "SETNUM", SEGMENT = 9, POINTS = 9$
SINPUT ITEM = "SETNUM", SEGMENT = 10, POINTS = 16$
SINPUT ITEM = "SETNUM", SEGMENT = 11, POINTS = 46$
SINPUT ITEM = "SETNUM", SEGMENT = 12, POINTS = 9$
SINPUT ITEM = "SETNUM", SEGMENT = 13, POINTS = 9$
SINPUT ITEM = "POINT", POINT = 1, R = 3.3333, 1.0, 0$
SINPUT ITEM = "POINT", POINT = 2, R = 8.216, 1.0, 0$
SINPUT ITEM = "POINT", POINT = 3, R = 10.0, 1.0, 0$
SINPUT ITEM = "POINT", POINT = 4, R = 11.882, 0.7, 0$
SINPUT ITEM = "POINT", POINT = 5, R = 72.0, 0.7, 0$
SINPUT ITEM = "POINT", POINT = 6, R = 40.0, 60.0, 0$
SINPUT ITEM = "POINT", POINT = 7, R = 3.3333, 60.0, 0$
SINPUT ITEM = "POINT", POINT = 8, R = 50.0, 60.0, 0$
SINPUT ITEM = "POINT", POINT = 9, R = 72.0, 60.0, 0$
SINPUT ITEM = "POINT", POINT = 10, R = 72.0, 1.8, 0$
SINPUT ITEM = "POINT", POINT = 11, R = -1.8, 0.0, 0$
SINPUT ITEM = "POINT", POINT = 12, R = -56.6667, 0.0, 0$
SINPUT ITEM = "POINT", POINT = 13, R = 0.0, 0.0, 0$
SINPUT ITEM = "POINT", POINT = 14, R = 9.51457, 1.8, 0$
SINPUT ITEM = "POINT", POINT = 15, R = 11.549, 1.8, 0$
SINPUT ITEM = "CONICUR",CURPTS=34,TYPE="CIRCLE",RADIUS=6.056,
  ANGLE=56.60428394,90.0$
SINPUT ITEM = "TRANS",POINTS=34,ORIGIN=3.3333,-5.056,0,
  COSINES=-1,0,0, 0,1,0, 0,0,-1,COREOUT=1$
SINPUT ITEM = "LINE", R1 = 1, R2 = 2, POINTS = -1$
SINPUT ITEM = "CURDIST",POINTSI=35,DISTYP="BOTH",RELATIV="NO",
  POINTS=35,SPACE=0.1,0.05,COREOUT=2$
SINPUT ITEM = "LINE", R1 = 2, R2 = 3, POINTS = -2, COREOUT= 3 S
SINPUT ITEM = "CONICUR",CURPTS=37,TYPE="CIRCLE",RADIUS=6.056,
   ANGLE=90.0,75.1802273$
SINPUT ITEM = "TRANS",POINTS=37,ORIGIN=10.0,-5.056,0,
   COSINES=1,0,0, 0,1,0, 0,0,1, COREOUT=4$
SINPUT ITEM = "CONICUR",CURPTS=4,TYPE="CIRCLE",RADIUS=6.056,
   ANGLE=75.1802273,71.89458975$
SINPUT ITEM = "TRANS",POINTS=4,ORIGIN=10.0,-5.056,0,
   COSINES=1,0,0, 0,1,0, 0,0,1, COREOUT=5$
SINPUT ITEM = "LINE", R1 = 4, R2 = 5, POINTS = -3$
SINPUT ITEM = "CURDIST",POINTSI=25,DISTYP="TANH",RELATIV="NO",
  POINTS=25,SPACE=0.10,COREOUT=6$
SINPUT ITEM = "CURRENT",COREIN=1,POINTS=34$
SINPUT ITEM = "INSERT",COREIN=2,POINTSI=35,START=34,POINTS=68$
SINPUT ITEM = "INSERT",COREIN=3,POINTSI=10,START=68,POINTS=77$
SINPUT ITEM = "INSERT",COREIN=4,POINTSI=37,START=77,POINTS=113$
SINPUT ITEM = "INSERT",COREIN=5,POINTSI=4,START=113,POINTS=116$
SINPUT ITEM = "INSERT",COREIN=6,POINTSI=25,START=116,POINTS=140,
  COREOUT=10$
SINPUT ITEM = "CURRENT",POINTS=250,
                    VALUES = 8.216000, 1.000000, 0.000000,
                             8.229360, 1.000000, 0.024381,
                             8.242720, 1.000000, 0.034066,
                             8.256080, 1.000000, 0.041321,
                             8.269440, 1.000000, 0.047311,
                             8.282800, 1.000000, 0.052488,
                             8.296160, 1.000000, 0.057082,
                             8.309519, 1.000000, 0.061233,
                             8.322879, 1.000000, 0.065029,
```

8.336239, 1.000000, 0.068532,
8.349599, 1.000000, 0.071789,
8.362959, 1.000000, 0.074834,
8.376319, 1.000000, 0.077693,
8.389679, 1.000000, 0.080387,
8.403039, 1.000000, 0.082935,
8.416399, 1.000000, 0.085349,
8.429759, 1.000000, 0.087643,
8.443119, 1.000000, 0.089826,
8.456479, 1.000000, 0.091906,
8.469838, 1.000000, 0.093892,
8.483198, 1.000000, 0.095790,
8.496558, 1.000000, 0.097605,
8.509918, 1.000000, 0.099343,
8.523278, 1.000000, 0.101008,
8.536638, 1.000000, 0.102604,
8.549998, 1.000000, 0.104136,
8.563358, 1.000000, 0.105605,
8.576718, 1.000000, 0.107016,
8.590078, 1.000000, 0.108370,
8.603438, 1.000000, 0.109671,
8.616798, 1.000000, 0.110921,
8.630158, 1.000000, 0.112121,
8.643517, 1.000000, 0.113275,
8.656877, 1.000000, 0.114382,
8.670237, 1.000000, 0.115447,
8.683597, 1.000000, 0.116468,
8.696957, 1.000000, 0.117450,
8.710317, 1.000000, 0.118391,
8.723677, 1.000000, 0.119295,
8.737037, 1.000000, 0.120162,
8.750397, 1.000000, 0.120993,
8.763757, 1.000000, 0.121790,
8.777117, 1.000000, 0.122553,
8.790477, 1.000000, 0.123283,
8.803836, 1.000000, 0.123982,
8.817196, 1.000000, 0.124649,
8.830556, 1.000000, 0.125287,
8.843916, 1.000000, 0.125895,
8.857276, 1.000000, 0.126474,
8.870636, 1.000000, 0.127026,
8.883996, 1.000000, 0.127551,
8.897356, 1.000000, 0.128049,
8.910716, 1.000000, 0.128521,
8.924076, 1.000000, 0.128968,
8.937436, 1.000000, 0.129391,
8.950796, 1.000000, 0.129789,
8.964156, 1.000000, 0.130163,
8.977515, 1.000000, 0.130514,
8.990875, 1.000000, 0.130843,
9.004235, 1.000000, 0.131150,
9.017595, 1.000000, 0.131434,
9.030955, 1.000000, 0.131698,
9.044315, 1.000000, 0.131941,
9.057675, 1.000000, 0.132163,
9.071035, 1.000000, 0.132365,
9.084395, 1.000000, 0.132548,
9.097755, 1.000000, 0.132711,
9.111115, 1.000000, 0.132855,
9.124475, 1.000000, 0.132981,
9.137834, 1.000000, 0.133089,
9.151194, 1.000000, 0.133179,
9.164554, 1.000000, 0.133251,
9.177914, 1.000000, 0.133306,
9.191274, 1.000000, 0.133344,
9.204634, 1.000000, 0.133365,

9.217994, 1.000000, 0.133370.
9.231354, 1.000000, 0.133359.
9.244714, 1.000000, 0.133332.
9.258074, 1.000000, 0.133290.
9.271434, 1.000000, 0.133232.
9.284794, 1.000000, 0.133159.
9.298154, 1.000000, 0.133072.
9.311513, 1.000000, 0.132969.
9.324873, 1.000000, 0.132853.
9.338233, 1.000000, 0.132723.
9.351593, 1.000000, 0.132578.
9.364953, 1.000000, 0.132420.
9.378313, 1.000000, 0.132249.
9.391673, 1.000000, 0.132064.
9.405033, 1.000000, 0.131866.
9.418393, 1.000000, 0.131656.
9.431753, 1.000000, 0.131433.
9.445113, 1.000000, 0.131197.
9.458473, 1.000000, 0.130950.
9.471832, 1.000000, 0.130690.
9.485192, 1.000000, 0.130418.
9.498552, 1.000000, 0.130135.
9.511912, 1.000000, 0.129840.
9.525272, 1.000000, 0.129534.
9.538632, 1.000000, 0.129216.
9.551992, 1.000000, 0.128888.
9.565352, 1.000000, 0.128549.
9.578712, 1.000000, 0.128199.
9.592072, 1.000000, 0.127838.
9.605432, 1.000000, 0.127467.
9.618792, 1.000000, 0.127086.
9.632152, 1.000000, 0.126694.
9.645511, 1.000000, 0.126293.
9.658871, 1.000000, 0.125881.
9.672231, 1.000000, 0.125460.
9.685591, 1.000000, 0.125030.
9.698951, 1.000000, 0.124590.
9.712311, 1.000000, 0.124140.
9.725671, 1.000000, 0.123682.
9.739031, 1.000000, 0.123214.
9.752391, 1.000000, 0.122737.
9.765751, 1.000000, 0.122252.
9.779111, 1.000000, 0.121758.
9.792471, 1.000000, 0.121255.
9.805830, 1.000000, 0.120744.
9.819190, 1.000000, 0.120224.
9.832550, 1.000000, 0.119696.
9.845910, 1.000000, 0.119160.
9.859270, 1.000000, 0.118615.
9.872630, 1.000000, 0.118063.
9.885990, 1.000000, 0.117503.
9.899350, 1.000000, 0.116935.
9.912710, 1.000000, 0.116359.
9.926070, 1.000000, 0.115776.
9.939430, 1.000000, 0.115186.
9.952790, 1.000000, 0.114587.
9.966149, 1.000000, 0.113982.
9.979509, 1.000000, 0.113369.
9.992869, 1.000000, 0.112749.
10.006229, 1.000000, 0.112122.
10.019589, 1.000000, 0.111488.
10.032949, 0.999985, 0.110849.
10.046309, 0.999941, 0.110205.
10.059669, 0.999867, 0.109558.
10.073029, 0.999764, 0.108907.
10.086389, 0.999632, 0.108252.

10.099749, 0.999469, 0.107593,
10.113109, 0.999278, 0.106931,
10.126469, 0.999057, 0.106264,
10.139828, 0.998806, 0.105594,
10.153188, 0.998526, 0.104920,
10.166548, 0.998216, 0.104242,
10.179908, 0.997877, 0.103560,
10.193268, 0.997509, 0.102875,
10.206628, 0.997111, 0.102186,
10.219988, 0.996683, 0.101492,
10.233348, 0.996226, 0.100796,
10.246708, 0.995739, 0.100095,
10.260068, 0.995223, 0.099390,
10.273428, 0.994677, 0.098682,
10.286788, 0.994102, 0.097969,
10.300147, 0.993497, 0.097253,
10.313507, 0.992863, 0.096533,
10.326867, 0.992199, 0.095809,
10.340227, 0.991505, 0.095080,
10.353587, 0.990782, 0.094348,
10.366947, 0.990029, 0.093612,
10.380307, 0.989247, 0.092872,
10.393667, 0.988435, 0.092127,
10.407027, 0.987593, 0.091379,
10.420387, 0.986721, 0.090626,
10.433747, 0.985820, 0.089869,
10.447107, 0.984890, 0.089108,
10.460467, 0.983929, 0.088342,
10.473826, 0.982939, 0.087572,
10.487186, 0.981919, 0.086798,
10.500546, 0.980870, 0.086019,
10.513906, 0.979790, 0.085236,
10.527266, 0.978681, 0.084448,
10.540626, 0.977542, 0.083655,
10.553986, 0.976373, 0.082858,
10.567346, 0.975175, 0.082056,
10.580706, 0.973947, 0.081249,
10.594066, 0.972688, 0.080437,
10.607426, 0.971400, 0.079621,
10.620786, 0.970082, 0.078799,
10.634145, 0.968734, 0.077972,
10.647505, 0.967356, 0.077140,
10.660865, 0.965948, 0.076303,
10.674225, 0.964511, 0.075461,
10.687585, 0.963043, 0.074613,
10.700945, 0.961545, 0.073760,
10.714305, 0.960017, 0.072901,
10.727665, 0.958459, 0.072036,
10.741025, 0.956871, 0.071166,
10.754385, 0.955253, 0.070290,
10.767745, 0.953605, 0.069408,
10.781105, 0.951926, 0.068520,
10.794465, 0.950218, 0.067626,
10.807824, 0.948479, 0.066726,
10.821184, 0.946710, 0.065820,
10.834544, 0.944910, 0.064907,
10.847904, 0.943081, 0.063988,
10.861264, 0.941221, 0.063062,
10.874624, 0.939330, 0.062129,
10.887984, 0.937409, 0.061190,
10.901344, 0.935458, 0.060244,
10.914704, 0.933477, 0.059290,
10.928064, 0.931465, 0.058330,
10.941424, 0.929422, 0.057362,
10.954784, 0.927349, 0.056387,
10.968143, 0.925245, 0.055405,

```
                                10.981503, 0.923111, 0.054415,
                                10.994863, 0.920946, 0.053417,
                                11.008223, 0.918751, 0.052412,
                                11.021583, 0.916525, 0.051398,
                                11.034943, 0.914268, 0.050376,
                                11.048303, 0.911980, 0.049346,
                                11.061663, 0.909662, 0.048308,
                                11.075023, 0.907312, 0.047261,
                                11.088383, 0.904932, 0.046206,
                                11.101743, 0.902521, 0.045141,
                                11.115103, 0.900079, 0.044068,
                                11.128463, 0.897606, 0.042986,
                                11.141822, 0.895102, 0.041894,
                                11.155182, 0.892567, 0.040793,
                                11.168542, 0.890000, 0.039683,
                                11.181902, 0.887403, 0.038563,
                                11.195262, 0.884774, 0.037433,
                                11.208622, 0.882115, 0.036293,
                                11.221982, 0.879424, 0.035143,
                                11.235342, 0.876701, 0.033983,
                                11.248702, 0.873948, 0.032812,
                                11.262062, 0.871162, 0.031630,
                                11.275422, 0.868346, 0.030438,
                                11.288782, 0.865498, 0.029234,
                                11.302141, 0.862618, 0.028020,
                                11.315501, 0.859707, 0.026794,
                                11.328861, 0.856764, 0.025556,
                                11.342221, 0.853790, 0.024307,
                                11.355581, 0.850784, 0.023046,
                                11.368941, 0.847746, 0.021773,
                                11.382301, 0.844676, 0.020488,
                                11.395661, 0.841575, 0.019190,
                                11.409021, 0.838441, 0.017879,
                                11.422381, 0.835276, 0.016555,
                                11.435741, 0.832078, 0.015219,
                                11.449101, 0.828849, 0.013869,
                                11.462461, 0.825587, 0.012506,
                                11.475820, 0.822293, 0.011129,
                                11.489180, 0.818967, 0.009738,
                                11.502540, 0.815609, 0.008332,
                                11.515900, 0.812218, 0.006913,
                                11.529260, 0.808795, 0.005479,
                                11.542620, 0.805339, 0.004030S
SINPUT ITEM = "CURDIST",POINTSI=250,POINTS=46,
   DISTYP="TANH",SPACE=.05,RELATIV="NO",COREOUT=9S
SINPUT ITEM = "SCALE",POINTS=46,SCALE=1,1,-1,COREOUT=8S
SINPUT ITEM = "CURRENT",COREIN=10,POINTS=140S
SINPUT ITEM = "INSERT",COREIN=8,POINTSI=46,START=68,POINTS=140,
  COREOUT=7S
SINPUT ITEM = "CURRENT",COREIN=10,POINTS=140S
SINPUT ITEM = "INSERT",COREIN=9,POINTSI=46,START=68,POINTS=140,
  COREOUT=6S
SINPUT ITEM = "CURRENT",POINTS=250,
                      VALUES = 9.428000, 1.700000, 0.000000,
                               9.436501, 1.700000, 0.015514,
                               9.445002, 1.700000, 0.021676,
                               9.453503, 1.700000, 0.026293,
                               9.462004, 1.700000, 0.030104,
                               9.470505, 1.700000, 0.033398,
                               9.479006, 1.700000, 0.036322,
                               9.487507, 1.700000, 0.038963,
                               9.496008, 1.700000, 0.041378,
                               9.504509, 1.700000, 0.043608,
                               9.513010, 1.700000, 0.045680,
                               9.521511, 1.700000, 0.047617,
                               9.530012, 1.700000, 0.049437,
```

9.538513, 1.700000, 0.051151,
9.547014, 1.700000, 0.052772,
9.555515, 1.700000, 0.054308,
9.564016, 1.700000, 0.055768,
9.572517, 1.700000, 0.057157,
9.581018, 1.700000, 0.058480,
9.589519, 1.700000, 0.059744,
9.598020, 1.700000, 0.060952,
9.606521, 1.700000, 0.062107,
9.615022, 1.700000, 0.063212,
9.623523, 1.700000, 0.064272,
9.632024, 1.700000, 0.065288,
9.640525, 1.700000, 0.066262,
9.649026, 1.700000, 0.067197,
9.657527, 1.700000, 0.068095,
9.666028, 1.700000, 0.068957,
9.674529, 1.700000, 0.069785,
9.683030, 1.700000, 0.070580,
9.691531, 1.700000, 0.071344,
9.700032, 1.700000, 0.072077,
9.708533, 1.700000, 0.072782,
9.717034, 1.700000, 0.073459,
9.725535, 1.700000, 0.074110,
9.734036, 1.700000, 0.074734,
9.742537, 1.700000, 0.075333,
9.751038, 1.700000, 0.075908,
9.759539, 1.700000, 0.076460,
9.768040, 1.700000, 0.076989,
9.776541, 1.700000, 0.077496,
9.785042, 1.700000, 0.077981,
9.793543, 1.700000, 0.078446,
9.802044, 1.700000, 0.078890,
9.810545, 1.700000, 0.079315,
9.819046, 1.700000, 0.079721,
9.827547, 1.700000, 0.080108,
9.836048, 1.700000, 0.080477,
9.844549, 1.700000, 0.080828,
9.853050, 1.700000, 0.081161,
9.861551, 1.700000, 0.081478,
9.870052, 1.700000, 0.081779,
9.878553, 1.700000, 0.082063,
9.887054, 1.700000, 0.082332,
9.895555, 1.700000, 0.082585,
9.904056, 1.700000, 0.082824,
9.912557, 1.700000, 0.083047,
9.921058, 1.700000, 0.083256,
9.929559, 1.700000, 0.083451,
9.938060, 1.700000, 0.083633,
9.946561, 1.700000, 0.083800,
9.955062, 1.700000, 0.083955,
9.963563, 1.700000, 0.084096,
9.972064, 1.700000, 0.084225,
9.980565, 1.700000, 0.084341,
9.989066, 1.700000, 0.084445,
9.997567, 1.700000, 0.084537,
10.006068, 1.700000, 0.084617,
10.014569, 1.700000, 0.084685,
10.023070, 1.700000, 0.084742,
10.031571, 1.700000, 0.084788,
10.040072, 1.700000, 0.084823,
10.048573, 1.700000, 0.084847,
10.057074, 1.700000, 0.084861,
10.065575, 1.700000, 0.084864,
10.074076, 1.700000, 0.084857,
10.082577, 1.700000, 0.084840,
10.091078, 1.700000, 0.084813,

10.099579, 1.700000, 0.084776,
10.108080, 1.700000, 0.084730,
10.116581, 1.700000, 0.084674,
10.125082, 1.700000, 0.084609,
10.133583, 1.700000, 0.084535,
10.142084, 1.700000, 0.084452,
10.150585, 1.700000, 0.084360,
10.159086, 1.700000, 0.084260,
10.167587, 1.700000, 0.084151,
10.176088, 1.700000, 0.084033,
10.184589, 1.700000, 0.083907,
10.193090, 1.700000, 0.083774,
10.201591, 1.700000, 0.083632,
10.210092, 1.700000, 0.083482,
10.218593, 1.700000, 0.083324,
10.227094, 1.700000, 0.083159,
10.235595, 1.700000, 0.082986,
10.244096, 1.700000, 0.082806,
10.252597, 1.700000, 0.082618,
10.261098, 1.700000, 0.082423,
10.269599, 1.700000, 0.082221,
10.278100, 1.700000, 0.082012,
10.286601, 1.700000, 0.081796,
10.295102, 1.700000, 0.081574,
10.303603, 1.700000, 0.081344,
10.312104, 1.700000, 0.081108,
10.320605, 1.700000, 0.080865,
10.329106, 1.700000, 0.080616,
10.337607, 1.700000, 0.080361,
10.346108, 1.700000, 0.080099,
10.354609, 1.700000, 0.079831,
10.363110, 1.700000, 0.079557,
10.371611, 1.700000, 0.079277,
10.380112, 1.700000, 0.078991,
10.388613, 1.700000, 0.078699,
10.397114, 1.700000, 0.078402,
10.405615, 1.700000, 0.078099,
10.414116, 1.700000, 0.077790,
10.422617, 1.700000, 0.077475,
10.431118, 1.700000, 0.077155,
10.439619, 1.700000, 0.076830,
10.448120, 1.700000, 0.076499,
10.456621, 1.700000, 0.076163,
10.465122, 1.700000, 0.075822,
10.473623, 1.700000, 0.075476,
10.482124, 1.700000, 0.075124,
10.490625, 1.700000, 0.074768,
10.499126, 1.700000, 0.074407,
10.507627, 1.700000, 0.074040,
10.516128, 1.700000, 0.073669,
10.524629, 1.700000, 0.073293,
10.533130, 1.700000, 0.072913,
10.541631, 1.700000, 0.072527,
10.550132, 1.700000, 0.072137,
10.558633, 1.700000, 0.071743,
10.567134, 1.700000, 0.071344,
10.575635, 1.700000, 0.070941,
10.584136, 1.700000, 0.070533,
10.592637, 1.700000, 0.070121,
10.601138, 1.700000, 0.069704,
10.609639, 1.700000, 0.069283,
10.618140, 1.700000, 0.068858,
10.626641, 1.700000, 0.068429,
10.635142, 1.700000, 0.067996,
10.643643, 1.700000, 0.067559,
10.652144, 1.700000, 0.067117,

10.660645, 1.700000, 0.066672,
10.669146, 1.700000, 0.066223,
10.677647, 1.700000, 0.065769,
10.686148, 1.700000, 0.065312,
10.694649, 1.700000, 0.064851,
10.703150, 1.700000, 0.064386,
10.711651, 1.700000, 0.063918,
10.720152, 1.700000, 0.063445,
10.728653, 1.700000, 0.062969,
10.737154, 1.700000, 0.062489,
10.745655, 1.700000, 0.062006,
10.754156, 1.700000, 0.061519,
10.762657, 1.700000, 0.061028,
10.771158, 1.700000, 0.060534,
10.779659, 1.700000, 0.060037,
10.788160, 1.700000, 0.059535,
10.796661, 1.700000, 0.059031,
10.805162, 1.700000, 0.058523,
10.813663, 1.700000, 0.058011,
10.822164, 1.700000, 0.057496,
10.830665, 1.700000, 0.056978,
10.839166, 1.700000, 0.056456,
10.847667, 1.700000, 0.055932,
10.856168, 1.700000, 0.055403,
10.864669, 1.700000, 0.054872,
10.873170, 1.700000, 0.054337,
10.881671, 1.700000, 0.053799,
10.890172, 1.700000, 0.053258,
10.898673, 1.700000, 0.052713,
10.907174, 1.700000, 0.052165,
10.915675, 1.700000, 0.051614,
10.924176, 1.700000, 0.051060,
10.932677, 1.700000, 0.050503,
10.941178, 1.700000, 0.049943,
10.949679, 1.700000, 0.049379,
10.958180, 1.700000, 0.048813,
10.966681, 1.700000, 0.048243,
10.975182, 1.700000, 0.047670,
10.983683, 1.700000, 0.047095,
10.992184, 1.700000, 0.046516,
11.000685, 1.700000, 0.045934,
11.009186, 1.700000, 0.045349,
11.017687, 1.700000, 0.044760,
11.026188, 1.700000, 0.044169,
11.034689, 1.700000, 0.043575,
11.043190, 1.700000, 0.042978,
11.051691, 1.700000, 0.042378,
11.060192, 1.700000, 0.041774,
11.068693, 1.700000, 0.041168,
11.077194, 1.700000, 0.040559,
11.085695, 1.700000, 0.039946,
11.094196, 1.700000, 0.039331,
11.102697, 1.700000, 0.038713,
11.111198, 1.700000, 0.038091,
11.119699, 1.700000, 0.037467,
11.128200, 1.700000, 0.036840,
11.136701, 1.700000, 0.036209,
11.145202, 1.700000, 0.035576,
11.153703, 1.700000, 0.034939,
11.162204, 1.700000, 0.034300,
11.170705, 1.700000, 0.033657,
11.179206, 1.700000, 0.033011,
11.187707, 1.700000, 0.032363,
11.196208, 1.700000, 0.031711,
11.204709, 1.700000, 0.031056,
11.213210, 1.700000, 0.030398,

```
                                    11.221711, 1.700000, 0.029737,
                                    11.230212, 1.700000, 0.029073,
                                    11.238713, 1.700000, 0.028406,
                                    11.247214, 1.700000, 0.027736,
                                    11.255715, 1.700000, 0.027062,
                                    11.264216, 1.700000, 0.026386,
                                    11.272717, 1.700000, 0.025706,
                                    11.281218, 1.700000, 0.025023,
                                    11.289719, 1.700000, 0.024337,
                                    11.298220, 1.700000, 0.023648,
                                    11.306721, 1.700000, 0.022955,
                                    11.315222, 1.700000, 0.022260,
                                    11.323723, 1.700000, 0.021561,
                                    11.332224, 1.700000, 0.020858,
                                    11.340725, 1.700000, 0.020153,
                                    11.349226, 1.700000, 0.019444,
                                    11.357727, 1.700000, 0.018732,
                                    11.366228, 1.700000, 0.018016,
                                    11.374729, 1.700000, 0.017298,
                                    11.383230, 1.700000, 0.016575,
                                    11.391731, 1.700000, 0.015850,
                                    11.400232, 1.700000, 0.015121,
                                    11.408733, 1.700000, 0.014388,
                                    11.417234, 1.700000, 0.013652,
                                    11.425735, 1.700000, 0.012913,
                                    11.434236, 1.700000, 0.012170,
                                    11.442737, 1.700000, 0.011423,
                                    11.451238, 1.700000, 0.010673,
                                    11.459739, 1.700000, 0.009920,
                                    11.468240, 1.700000, 0.009162,
                                    11.476741, 1.700000, 0.008401,
                                    11.485242, 1.700000, 0.007637,
                                    11.493743, 1.700000, 0.006868,
                                    11.502244, 1.700000, 0.006096,
                                    11.510745, 1.700000, 0.005321,
                                    11.519246, 1.700000, 0.004541,
                                    11.527747, 1.700000, 0.003757,
                                    11.536248, 1.700000, 0.002970,
                                    11.544749, 1.700000, 0.002179$
SINPUT ITEM = "CURDIST",POINTSI=250,POINTS=46,
   DISTYP="TANH",SPACE=.0318,RELATIV="NO",COREOUT=3$
SINPUT ITEM = "SCALE",POINTS=46,SCALE=1.1,-1,COREOUT=5$
SINPUT ITEM = "LINE", R1 = 14, R2 = 15, POINTS = -11,
   DISTYP="TANH",SPACE=-1,RELATIV="NO",COREOUT=2 $
SINPUT ITEM="BOUNCUR",COREIN=3,POINTS=46$
SINPUT ITEM = "BOUNCUR",COREIN=2,POINTS=46$
SINPUT ITEM = "BLEND",POINTS=46,CURVES=2,COREOUT=1$
SINPUT ITEM = "BOUNCUR",COREIN=9,POINTS=46$
SINPUT ITEM = "BOUNCUR",COREIN=3,POINTS=46$
SINPUT ITEM = "BLEND",POINTS=46,CURVES=15$
SINPUT ITEM = "INSERT",COREIN=1,POINTSI=46,2,START=1,15,POINTS=46,16,
  COREOUT=4$
SINPUT ITEM = "BOUNCUR",COREIN=5,POINTS=46$
SINPUT ITEM = "BOUNCUR",COREIN=2,POINTS=46$
SINPUT ITEM = "BLEND",POINTS=46,CURVES=2,COREOUT=1$
SINPUT ITEM = "BOUNCUR",COREIN=8,POINTS=46$
SINPUT ITEM = "BOUNCUR",COREIN=5,POINTS=46$
SINPUT ITEM = "BLEND",POINTS=46,CURVES=15$
SINPUT ITEM = "INSERT",COREIN=1,POINTSI=46,2,START=1,15,POINTS=46,16,
  COREOUT=5$
SINPUT ITEM = "BOUNCUR",COREIN=6,POINTS=140$
SINPUT ITEM = "BOUNCUR",COREIN=7,POINTS=140$
SINPUT ITEM = "ROTATE",CURPTS=140,ANGPTS=10,ANGLE=-45,45,
  DISTANG="BOTH",SPACANG=0.05,0.05,AXCOS=1.0,0,NORCOS=0,1.0,
   FILEOUT=1$
SINPUT ITEM = "BOUNCUR",COREIN=6,POINTS=140$
```

```
SINPUT ITEM = "BOUNCUR",COREIN=7,POINTS=140$
SINPUT ITEM = "ROTATE",CURPTS=140,ANGPTS=10,ANGLE=45,135,
  DISTANG="BOTH",SPACANG=0.05,0.05,AXCOS=1,0,0,NORCOS=0,1,0,
  RMIN=-100,-100,-100, RMAX=100,100,100, FILEOUT=2$
SINPUT ITEM = "BOUNCUR",COREIN=6,POINTS=140$
SINPUT ITEM = "BOUNCUR",COREIN=7,POINTS=140$
SINPUT ITEM = "ROTATE",CURPTS=140,ANGPTS=10,ANGLE=135,225,
  DISTANG="BOTH",SPACANG=0.05,0.05,AXCOS=1,0,0,NORCOS=0,1,0,
  RMIN=-100,-100,-100, RMAX=100,100,100, FILEOUT=3$
SINPUT ITEM = "BOUNCUR",COREIN=6,POINTS=140$
SINPUT ITEM = "BOUNCUR",COREIN=7,POINTS=140$
SINPUT ITEM = "ROTATE",CURPTS=140,ANGPTS=10,ANGLE=225,315,
  DISTANG="BOTH",SPACANG=0.05,0.05,AXCOS=1,0,0,NORCOS=0,1,0,
  RMIN=-100,-100,-100, RMAX=100,100,100, FILEOUT=4$
SINPUT ITEM = "TRANS",COREIN=5,POINTS=46,16,
  COSINES=1,0,0, 0,.707,.707, 0,-.707,.707, FILEOUT=5$
SINPUT ITEM = "TRANS",COREIN=4,POINTS=46,16,
  COSINES=1,0,0, 0,.707,.707, 0,-.707,.707, FILEOUT=6$
SINPUT ITEM = "TRANS",COREIN=5,POINTS=46,16,
  COSINES=1,0,0, 0,.707,-.707,0,.707,.707, FILEOUT=7$
SINPUT ITEM = "TRANS",COREIN=4,POINTS=46,16,
  COSINES=1,0,0, 0,.707,-.707, 0,.707,.707, FILEOUT=8$
SINPUT ITEM = "TRANS",COREIN=5,POINTS=46,16,
  COSINES=1,0,0, 0,-.707,-.707,0,.707,-.707, FILEOUT=9$
SINPUT ITEM = "TRANS",COREIN=4,POINTS=46,16,
  COSINES=1,0,0, 0,-.707,-.707, 0,.707,-.707, FILEOUT=10$
SINPUT ITEM = "TRANS",COREIN=5,POINTS=46,16,
  COSINES=1,0,0, 0,-.707,.707, 0,-.707,-.707, FILEOUT=33$
SINPUT ITEM = "TRANS",COREIN=4,POINTS=46,16,
  COSINES=1,0,0, 0,-.707,.707, 0,-.707,-.707, FILEOUT=35$
SINPUT ITEM = "LINE", R1 = 14, R2 = 6, POINTS = -12 $
SINPUT ITEM = "CURDIST",POINTSI=9,DISTYP="TANH",RELATIV="NO",
  SPACE=0.1,POINTS=9,COREOUT=1$
SINPUT ITEM = "TRANS",COREIN=1,POINTS=9,
  COSINES=1,0,0, 0,.707,.707, 0,-.707,.707, FILEOUT=34$
SINPUT ITEM = "TRANS",COREIN=1,POINTS=9,
  COSINES=1,0,0, 0,.707,-.707, 0,.707,.707, FILEOUT=14$
SINPUT ITEM = "TRANS",COREIN=1,POINTS=9,
  COSINES=1,0,0, 0,-.707,-.707, 0,.707,-.707, FILEOUT=15$
SINPUT ITEM = "TRANS",COREIN=1,POINTS=9,
  COSINES=1,0,0, 0,-.707,.707, 0,-.707,-.707, FILEOUT=16$
SINPUT ITEM = "LINE", R1 = 15, R2 = 8, POINTS = -13 $
SINPUT ITEM = "CURDIST",POINTSI=9,DISTYP="TANH",RELATIV="NO",
  SPACE=0.1,POINTS=9,COREOUT=1$
SINPUT ITEM = "TRANS",COREIN=1,POINTS=9,
  COSINES=1,0,0, 0,.707,.707, 0,-.707,.707, FILEOUT=17$
SINPUT ITEM = "TRANS",COREIN=1,POINTS=9,
  COSINES=1,0,0, 0,.707,-.707, 0,.707,.707, FILEOUT=18$
SINPUT ITEM = "TRANS",COREIN=1,POINTS=9,
  COSINES=1,0,0, 0,-.707,-.707, 0,.707,-.707, FILEOUT=19$
SINPUT ITEM = "TRANS",COREIN=1,POINTS=9,
  COSINES=1,0,0, 0,-.707,.707, 0,-.707,-.707, FILEOUT=20$
SINPUT ITEM = "CONICUR",CURPTS=60,TYPE="CIRCLE",
  RADIUS=60.0,ANGLE=90.0,0.0$
SINPUT ITEM = "TRANS",POINTS=60,ORIGIN=3.3333,0,0,
  COSINES=-1,0,0, 0,1,0, 0,0,-1, COREOUT=1$
SINPUT ITEM = "LINE", R1 = 6, R2 = 7, POINTS = -4 $
SINPUT ITEM = "INSERT",COREIN=1,POINTSI=60,START=9,POINTS=68$
SINPUT ITEM = "CURDIST",POINTSI=68,DISTYP="TANH",RELATIV="NO",
  POINTS=68,SPACE=0.2$
SINPUT ITEM = "SWITCH",POINTS=68,REORDER="REVERSE1",COREOUT=1$
SINPUT ITEM = "LINE", R1 = 6, R2 = 8, POINTS = -5,
   DISTYP="BOTH",RELATIV="NO",SPACE=-2,-2,COREOUT=2$
SINPUT ITEM = "LINE", R1 = 8, R2 = 9, POINTS = -6 $
SINPUT ITEM = "CURDIST",POINTSI=28,DISTYP="TANH",RELATIV="NO",
   SPACE=0.2,POINTS=28,COREOUT=11$
```

```
SINPUT ITEM = "CURRENT",COREIN=1,POINTS=68$
SINPUT ITEM = "INSERT",COREIN=2,POINTSI=46,START=68,POINTS=113$
SINPUT ITEM = "INSERT",COREIN=11,POINTSI=28,START=113,POINTS=140,
   COREOUT=11$
SINPUT ITEM = "BOUNCUR",POINTS=140$
SINPUT ITEM = "ROTATE",CURPTS=140,ANGPTS=10,ANGLE=-45,45,
  DISTANG="BOTH",SPACANG=0.05,0.05,AXCOS=1,0,0,NORCOS=0,1,0,
  RMIN=-100,-100,-100, RMAX=100,100,100, VIEW=120,10,5000,
   FILEOUT=21 S
SINPUT ITEM = "LINE", R1 = 10, R2 = 9, POINTS = -7 S
SINPUT ITEM = "CURDIST",POINTSI=9,DISTYP="TANH",RELATIV="NO",
  POINTS=9,SPACE=0.1,COREOUT=12$
SINPUT ITEM = "LINE", R1 = 5, R2 = 10, POINTS = -8 S
SINPUT ITEM = "INSERT",COREIN=12,POINTSI=9,START=16,POINTS=24,
   COREOUT=14$
SINPUT ITEM = "BOUNCUR",POINTS=24$
SINPUT ITEM = "ROTATE",CURPTS=24,ANGPTS=10,ANGLE=-45,45,
  DISTANG="BOTH",SPACANG=0.05,0.05,AXCOS=1,0,0,NORCOS=0,1,0,
  RMIN=-100,-100,-100, RMAX=100,100,100,FILEOUT=22 S
SINPUT ITEM = "LINE", R1 = 11, R2 = 12, POINTS = -9 S
SINPUT ITEM = "CURDIST",POINTSI=9,DISTYP="TANH",RELATIV="NO",
  POINTS=9,SPACE=0.1,COREOUT=13$
SINPUT ITEM = "LINE", R1 = 13, R2 = 11, POINTS = -10 S
SINPUT ITEM = "INSERT",COREIN=13,POINTSI=9,START=16,POINTS=24,
   COREOUT=15$
SINPUT ITEM = "BOUNCUR",POINTS=24$
SINPUT ITEM = "ROTATE",CURPTS=24,ANGPTS=10,ANGLE=-45,45,
  DISTANG="BOTH",SPACANG=0.05,0.05,AXCOS=-1,0,0,NORCOS=0,1,0,
  RMIN=-100,-100,-100, RMAX=100,100,100 ,FILEOUT=23 S
SINPUT ITEM = "BOUNCUR",COREIN=11,POINTS=140$
SINPUT ITEM = "ROTATE",CURPTS=140,ANGPTS=10,ANGLE=45,135,
  DISTANG="BOTH",SPACANG=0.05,0.05,AXCOS=1,0,0,NORCOS=0,1,0,
  RMIN=-100,-100,-100, RMAX=100,100,100, FILEOUT=24 S
SINPUT ITEM = "BOUNCUR",COREIN=14,POINTS=24$
SINPUT ITEM = "ROTATE",CURPTS=24,ANGPTS=10,ANGLE=45,135,
  DISTANG="BOTH",SPACANG=0.05,0.05,AXCOS=1,0,0,NORCOS=0,1,0,
  RMIN=-100,-100,-100, RMAX=100,100,100, FILEOUT=25  S
SINPUT ITEM = "BOUNCUR",COREIN=15,POINTS=24$
SINPUT ITEM = "ROTATE",CURPTS=24,ANGPTS=10,ANGLE=45,135,
  DISTANG="BOTH",SPACANG=0.05,0.05,AXCOS=-1,0,0,NORCOS=0,1,0,
  RMIN=-100,-100,-100, RMAX=100,100,100, FILEOUT=26  S
SINPUT ITEM = "BOUNCUR",COREIN=11,POINTS=140$
SINPUT ITEM = "ROTATE",CURPTS=140,ANGPTS=10,ANGLE=135,225,
  DISTANG="BOTH",SPACANG=0.05,0.05,AXCOS=1,0,0,NORCOS=0,1,0,
  RMIN=-100,-100,-100, RMAX=100,100,100, FILEOUT=27 S
SINPUT ITEM = "BOUNCUR",COREIN=14,POINTS=24$
SINPUT ITEM = "ROTATE",CURPTS=24,ANGPTS=10,ANGLE=135,225,
  DISTANG="BOTH",SPACANG=0.05,0.05,AXCOS=1,0,0,NORCOS=0,1,0,
  RMIN=-100,-100,-100, RMAX=100,100,100, FILEOUT=28 S
SINPUT ITEM = "BOUNCUR",COREIN=15,POINTS=24$
SINPUT ITEM = "ROTATE",CURPTS=24,ANGPTS=10,ANGLE=135,225,
  DISTANG="BOTH",SPACANG=0.05,0.05,AXCOS=-1,0,0,NORCOS=0,1,0,
  RMIN=-100,-100,-100, RMAX=100,100,100, FILEOUT=29 S
SINPUT ITEM = "BOUNCUR",COREIN=11,POINTS=140$
SINPUT ITEM = "ROTATE",CURPTS=140,ANGPTS=10,ANGLE=225,315,
  DISTANG="BOTH",SPACANG=0.05,0.05,AXCOS=1,0,0,NORCOS=0,1,0,
  RMIN=-100,-100,-100, RMAX=100,100,100,FILEOUT=30 S
SINPUT ITEM = "BOUNCUR",COREIN=14,POINTS=24$
SINPUT ITEM = "ROTATE",CURPTS=24,ANGPTS=10,ANGLE=225,315,
  DISTANG="BOTH",SPACANG=0.05,0.05,AXCOS=1,0,0,NORCOS=0,1,0,
  RMIN=-100,-100,-100, RMAX=100,100,100, FILEOUT=31 S
SINPUT ITEM = "BOUNCUR",COREIN=15,POINTS=24$
SINPUT ITEM = "ROTATE",CURPTS=24,ANGPTS=10,ANGLE=225,315,
  DISTANG="BOTH",SPACANG=0.05,0.05,AXCOS=-1,0,0,NORCOS=0,1,0,
  RMIN=-100,-100,-100, RMAX=100,100,100, FILEOUT=32 S
SINPUT ITEM = "COMBINE", FILEIN=1,21,22,23,14,18,34,17,8,5,
```

```
   FILEOUT=40S
SINPUT ITEM="COMBINE",FILEIN=2,24,25,26,34,17,16,20,6,33,
   FILEOUT=50S
SINPUT ITEM="COMBINE",FILEIN=3,27,28,29,16,20,15,19,35,9,
   FILEOUT=60S
SINPUT ITEM="COMBINE",FILEIN=4,30,31,32,15,19,14,18,10,7,
   FILEOUT=70S
SINPUT ITEM = "END"S
```

GRID INPUT DATA

```
1.  $INPUT ITEM="STORE",KSTORE="FILE",FILE=72,ITMAX=1,TOL=1.0E-06,ACCPAR=.0,
      OUTER="NO", CHECK="NO"CHECK="NO"$
```

Line 1 stores the output of the field grid generation system on file 72. The maximum number of iterations will be one (for an algebraic grid) with a toleration parameter of 1.0E-06 and the acceleration parameter for the SOR point iteration set to 0.0 (for the algebraic grid). No extra layer of points will be placed outside the grid.

```
2.  $INPUT ITEM="INITIAL",CONTYP="INITIAL",BLEND="ARC","ARC","LINEAR",
      ALL="YES"$
```

Line 2 initializes the generation system with an algebraic grid and since the control function is also initial, only an algebraic grid will be developed. The blending functions in x, y, (I,J) will be based on interpolated arc lengths while, in the third dimension, z (K), the blending function will be a linear interpolation.

```
3.  $INPUT ITEM="BLOCK",SIZE=140,24,10$
```

Line 3 sets the maximum size of the first block at 140 x 24 x 10 (Figure 76).

```
4.  $INPUT ITEM="FILE",START= 1,1,1,END=140,1,10,CLASS="FIX",
      FILE=50,REWIND="YES"$
```

Line 4 reads in the body boundary (ogive-cylinnder-ogive with sting) and fixes these 140 x 1 x 10 points.

```
5.  $INPUT ITEM="FILE",START=1,24,1,END=140,24,10,CLASS="FIX",
      FILE=50,REWIND="NO"$
```

Line 5 reads in the outer boundary (C-type outer boundary) and sets those points as fixed.

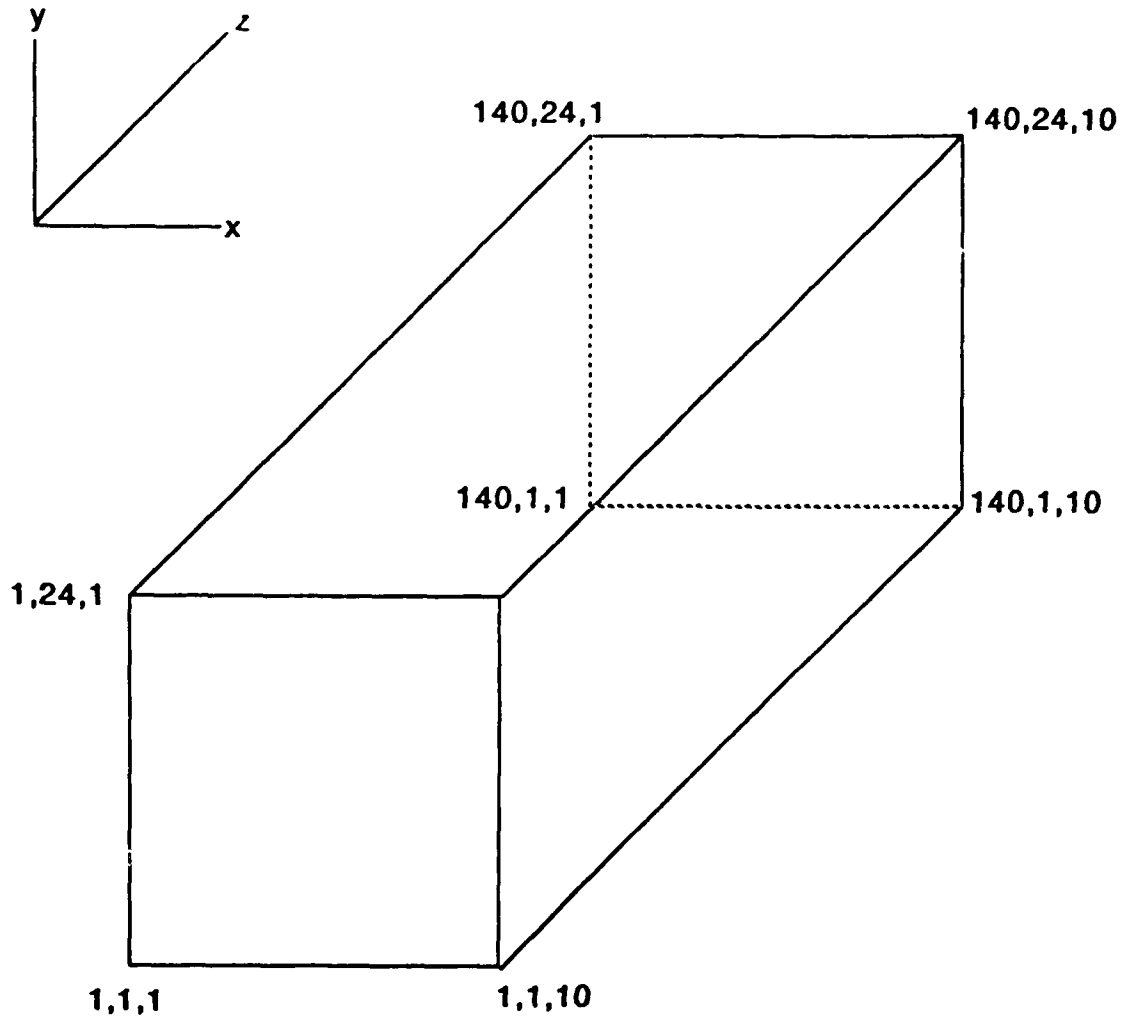


Figure 76. Single Block Format

```
6.  $INPUT ITEM="FILE",START=140,1,1,END=140,24,10,CLASS="FIX",
      FILE=50,REWIND="NO"$
```

Line 6 reads in the back boundary and sets those points as fixed.

```
7.  $INPUT ITEM="FILE",START=1,1,1,END=1,24,10,CLASS="FIX",
      FILE=50,REWIND="NO"$
```

Line 7 reads in the front stagnation line boundary and sets those points as fixed.

```
8.  $INPUT ITEM="FILE",START=68,16,1,END=68,24,1,FILE=50,
      REWIND="NO"$
9.  $INPUT ITEM="FILE",START=113,16,1,END=113,24,1,FILE=50,
      REWIND="NO"$
```

Lines 8 and 9 read in the leading and trailing edge controlling line boundary from the fin tip to the outer boundary for the left side of Block 1.

```
10. $INPUT ITEM="FILE",START=68,16,10,END=68,24,10,FILE=50,
      REWIND="NO"$
11. $INPUT ITEM="FILE",START=113,16,10,END=113,24,10,FILE=50,
      REWIND="NO"$
```

Lines 10 and 11 read in the leading and trailing edge controlling line boundary for the right side of Block 1.

```
12. $INPUT ITEM="FILE",START=68,1,1,END=113,16,1,CLASS="FIX",
      FILE=50,REWIND="NO"$
13. $INPUT ITEM="FILE",START=68,1,10,END=113,16,10,CLASS="FIX",
      FILE=50,REWIND="NO"$
```

Lines 12 and 13 read in the fin solid boundary on the left and right side of Block 1 and set those points to fixed.

```
14.  $INPUT ITEM="INTERP",BLEND="ARC","ARC",START=1,1,1,END=68,24,1$
```

Line 14 interpolates, based on arc lengths, the portion of the grid from 1,1,1 to 68, 24, 1 that was not specified by the surface grid generation system.

```
15.  $INPUT ITEM="INTERP",BLEND="ARC","ARC",START=68,16,1,END=113,24,1$
16.  $INPUT ITEM="INTERP",BLEND="ARC","ARC",START=113,1,1,END=140,24,1$
17.  $INPUT ITEM="INTERP",BLEND="ARC","ARC",START=1,1,10,END=68,24,10$
18.  $INPUT ITEM="INTERP",BLEND="ARC","ARC",START=68,16,10,END=113,24,10$
19.  $INPUT ITEM="INTERP",BLEND="ARC","ARC",START=113,1,10,END=140,24,10$
```

Lines 15-19 interpolate for the rest of the unspecified grid boundaries based on arc lengths (Figure 77).

```
20.  $INPUT ITEM="FIX",START=68,1,1,END=113,16,1$
21.  $INPUT ITEM="FIX",START=68,1,10,END=113,16,10$
```

Lines 20 and 21 fix the points on the fin surfaces for Block 1.

```
22.  $INPUT ITEM="BLOCK",SIZE=140,24,10$
```

Line 22 sets the maximum size of the second block at 140 x 24 x 10.

```
23.  $INPUT ITEM="FILE",START=1,1,1,END=140,1,10,CLASS="FIX",
       FILE=60,REWIND="YES"$
24.  $INPUT ITEM="FILE",START=1,24,1,END=140,24,10,CLASS="FIX",
       FILE=60,REWIND="NO"$
25.  $INPUT ITEM="FILE",START=140,1,1,END=140,24,10, CLASS="FIX",
       FILE=60,REWIND="NO"$
26.  $INPUT ITEM="FILE",START=1,1,1,END= 1,24,10,CLASS="FIX",
       FILE=60,REWIND="NO"$
27.  $INPUT ITEM="FILE",START=68,16,1,END=68,24,1,FILE=60,
       REWIND="NO"$
28.  $INPUT ITEM="FILE",START=113,16,1,END=113,24,1,FILE=60,
       REWIND="NO"$
29.  $INPUT ITEM="FILE",START=68,16,10,END=68,24,10,FILE=60,
       REWIND="NO"$
30.  $INPUT ITEM="FILE",START=113,16,10,END=113,24,10,FILE=60,
```

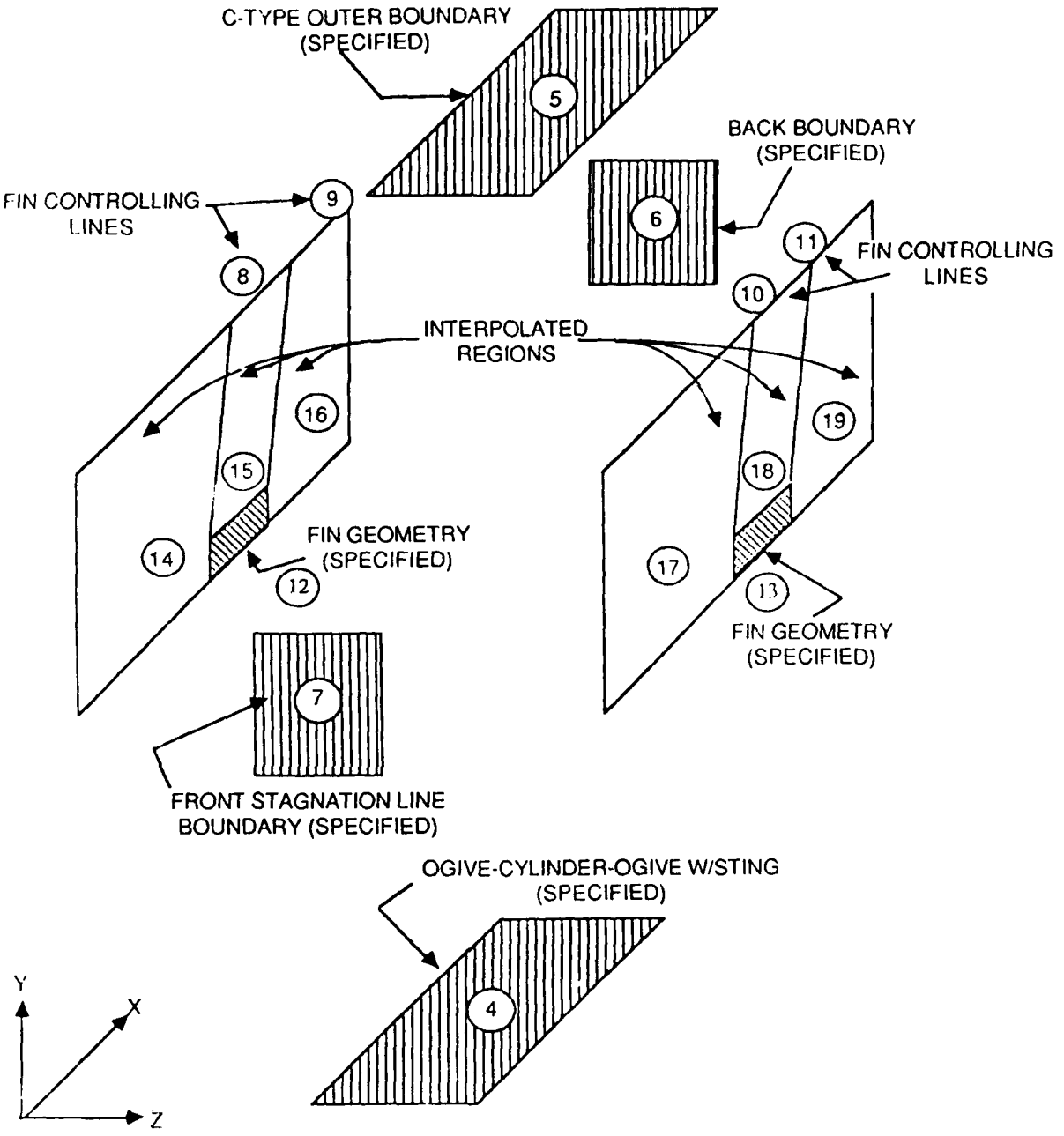


Figure 77. Single Block Format (Exploded View)

```
      REWIND="NO"$
31.  $INPUT ITEM="FILE",START=68,1,1,END=113,16,1,CLASS="FIX",
       FILE=60,REWIND="NO"$
32.  $INPUT ITEM="FILE",START=68,1,10,END=113,16,10,CLASS="FIX",
       FILE=60,REWIND="NO"$
```

Lines 23-32 are analagous to lines 4-13. The boundaries of Block 2 are read in from output files.

```
33.  $INPUT ITEM="INTERP",BLEND="ARC","ARC",START=1,1,1,END=68,24,1$
34.  $INPUT ITEM="INTERP",BLEND="ARC","ARC",START=68,16,1,END=113,24,1$
35.  $INPUT ITEM="INTERP",BLEND="ARC","ARC",START=113,1,1,END=140,24,1$
36.  $INPUT ITEM="INTERP",BLEND="ARC","ARC",START=1,1,10,END=68,24,10$
37.  $INPUT ITEM="INTERP",BLEND="ARC","ARC",START=68,16,10,END=113,24,10$
38.  $INPUT ITEM="INTERP",BLEND="ARC","ARC",START=113,1,10,END=140,24,10$
```

Lines 33-38 are analagous to lines 4-19 and interpolate the unspecified grid boundaries for Block 2.

```
39.  $INPUT ITEM="FIX",START=68,1,1,END=113,16,1$
40.  $INPUT ITEM="FIX",START=68,1,10,END=113,16,10$
```

Lines 39 and 40 fix the points on the fin surfaces for Block 2.

```
41.  $INPUT ITEM="BLOCK",SIZE=140,24,10$
```

Line 41 sets the maximum size of the third block at 140 x 24 x 10.

```
42.  $INPUT ITEM="FILE",START=1,1,1,END=140,1,10,CLASS="FIX",
       FILE=70,REWIND="YES"$
43.  $INPUT ITEM="FILE",START=1,24,1,END=140,24,10,CLASS="FIX",
       FILE=70,REWIND="NO"$
44.  $INPUT ITEM="FILE",START=140,1,1,END=140,24,10,CLASS="FIX",
       FILE=70,REWIND="NO"$
45.  $INPUT ITEM="FILE",START=1,1,1,END=1,24,10,CLASS="FIX",
       FILE=70,REWIND="NO"$
46.  $INPUT ITEM="FILE",START=68,16,1,END=68,24,1,FILE=70,
```

```
      REWIND="NO"$
47.  $INPUT ITEM="FILE",START=113,16,1,END=113,24,1,FILE=70,
      REWIND="NO"$
48.  $INPUT ITEM="FILE",START=68,16,10,END=68,24,10,FILE=70,
      REWIND="NO"$
49.  $INPUT ITEM="FILE",START=113,16,10,END=113,24,10,FILE=70,
      REWIND="NO"$
50.  $INPUT ITEM="FILE",START=68,1,1,END=113,16,1,CLASS="FIX",
      FILE=70,REWIND="NO"$
51.  $INPUT ITEM="FILE",START=68,1,10,END=113,16,10,CLASS="FIX",
      FILE=70,REWIND="NO"$
```

Lines 42-51 read in the Block 3 boundaries from output files.

```
52.  $INPUT ITEM="INTERP",BLEND="ARC","ARC",START=1,1,1,END=68,24,1$
53.  $INPUT ITEM="INTERP",BLEND="ARC","ARC",START=68,16,1,END=113,24,1$
54.  $INPUT ITEM="INTERP",BLEND="ARC","ARC",START=113,1,1,END=140,24,1$
55.  $INPUT ITEM="INTERP",BLEND="ARC","ARC",START=1,1,10,END=68,24,10$
56.  $INPUT ITEM="INTERP",BLEND="ARC","ARC",START=68,16,10,END=113,24,10$
57.  $INPUT ITEM="INTERP",BLEND="ARC","ARC",START=113,1,10,END=140,24,10$
```

Lines 52-57 interpolate the unspecified grid boundaries for Block 3.

```
58.  $INPUT ITEM="FIX",START=68,1,1,END=113,16,1$
59.  $INPUT ITEM="FIX",START=68,1,10,END=113,16,10$
```

Lines 58 and 59 fix the points on the fin surfaces for Block 3.

```
60.  $INPUT ITEM="BLOCK".SIZE=140,24,10$
```

Line 60 set the maximum size of the fourth block at 140 x 24 x 10.

```
61.  $INPUT ITEM="FILE",START=1,1,1,END=140,1,10,CLASS="FIX",
      FILE=80,REWIND="YES"$
62.  $INPUT ITEM="FILE",START=1,24,1,END=140,24,10,CLASS="FIX",
      FILE=80,REWIND="NO"$
```

```
63.  $INPUT ITEM="FILE",START=140,1,1,END=140,24,10,CLASS="FIX",
       FILE=80,REWIND="NO"$
64.  $INPUT ITEM="FILE",START=1,1,1,END=1,24,10,CLASS="FIX",
       FILE=80,REWIND="NO"$
65.  $INPUT ITEM="FILE",START=68,16,1,END=68,24,1,FILE=80,
       REWIND="NO"$
66.  $INPUT ITEM="FILE",START=113,16,1,END=113,24,1,FILE=80,
       REWIND="NO"$


67.  $INPUT ITEM="FILE",START=68,16,10,END=68,24,10,FILE=80,
       REWIND="NO"$
68.  $INPUT ITEM="FILE",START=113,16,10,END=113,24,10,FILE=80,
       REWIND="NO"$
69.  $INPUT ITEM="FILE",START=68,1,1,END=113,16,1,CLASS="FIX",
       FILE=80,REWIND="NO"$
70.  $INPUT ITEM="FILE",START=68,1,10,END=113,16,10,CLASS="FIX",
       FILE=80,REWIND="NO"$
```

Lines 61-70 read in the Block 4 boundaries from output files.

```
71.  $INPUT ITEM="INTERP",BLEND="ARC","ARC",START= 1,1,1,END=68,24,1$
72.  $INPUT ITEM="INTERP",BLEND="ARC","ARC",START=68,16,1,END=113,24,1$
73.  $INPUT ITEM="INTERP",BLEND="ARC","ARC",START=113,1,1,END=140,24,1$
74.  $INPUT ITEM="INTERP",BLEND="ARC","ARC",START=1,1,10,END=68,24,10$
75.  $INPUT ITEM="INTERP",BLEND="ARC","ARC",START=68,16,10,END=113,24,10$
76.  $INPUT ITEM="INTERP",BLEND="ARC","ARC",START=113,1,10,END=140,24,10$
```

Lines 71-76 interpolate the unspecified grid boundaries for Block 4.
Lines 22-40 read in and interpolate for all of Block 2.
Lines 41-59 read in and interpolate for all of Block 3.
Lines 60-78 read in and interpolate for all of Block 4.

```
77.  $INPUT ITEM="FIX",START=68,1,1,END=113,16,1$
78.  $INPUT ITEM="FIX",START=68,1,10,END=113,16,10$
```

Lines 77 and 78 fix the points on the fin surfaces for Block 4.

```
79.  $INPUT ITEM="CUT",BLOCK=1,START=1,1,1,END=68,24,1
       IBLOCK=4,ISTART=1,1,10,IEND=68,24,10$
80.  $INPUT ITEM="CUT",BLOCK=1,START=68, 16,1,END=113,24,1,
       IBLOCK=4,ISTART=68,16,10, IEND=113,24,10$
81.  $INPUT ITEM="CUT",BLOCK=1,START=113,1,1,END=140,24,1
       IBLOCK=2,ISTART=113,1,10,IEND=140,24,10$
```

Lines 79 through 81 defined the points which are common to Block 1 and Block 4.

```
82.  $INPUT ITEM="CUT",BLOCK=1,START=1,1,10,END=68,24,10,
       IBLOCK=2,ISTART=1,1,1,IEND=68,24,1$
83.  $INPUT ITEM="CUT",BLOCK=1,START=68,16,10,END=113,24,10,
       IBLOCK=2,ISTART=68,16,1,IEND=113,24,1$
84.  $INPUT ITEM="CUT",BLOCK=2,START=113,1,10,END=140,24,10,
       IBLOCK=3,ISTART=113,1,1,IEND=140,24,1$
```

Lines 82 through 84 define the points which are common to Block 1 and Block 2.

```
85.  $INPUT ITEM="CUT",BLOCK=2,START=1,1,10,END=68,24,10,
       IBLOCK=3,ISTART=1,1,1,IEND=68,24,1$
86.  $INPUT ITEM="CUT",BLOCK=2,START=68,16,10,END=113,24,10,
       IBLOCK=3,ISTART=68,16,1,IEND=113,24,1$
87.  $INPUT ITEM="CUT",BLOCK=3,START=113,1,10,END=140,24,10,
       IBLOCK=3,ISTART=113,1,1,IEND=140,24,1$
```

Lines 85 through 87 define the points which are common to Block 2 and Block 3.

```
88.  $INPUT ITEM="CUT",BLOCK=3,START=1,1,10,END=68,24,10,
       IBLOCK=4,ISTART=1,1,1,IEND=68,24,1$
89.  $INPUT ITEM="CUT",BLOCK=3,START=68,16,10,END=113,24,10,
       IBLOCK=4,ISTART=68,16,1,IEND=113,24,1$
90.  $INPUT ITEM="CUT",BLOCK=3,START=113,1,10,END=140,24,10,
       IBLOCK=4,ISTART=113,1,1,IEND=140,24,1$
```

Lines 88 through 90 define the points which are common to Block 3 and Block 4.

91. $INPUT ITEM = "END" $

Line 91 terminates input to the field grid generation system.

92. $OUTPUT ITEM = "ERROR", BLKERR="YES" $

Line 92 causes the maximum iteration error in each block to be printed at each iteration.

93. $OUTPUT ITEM = "END" $

Line 93 terminates the print-out input.

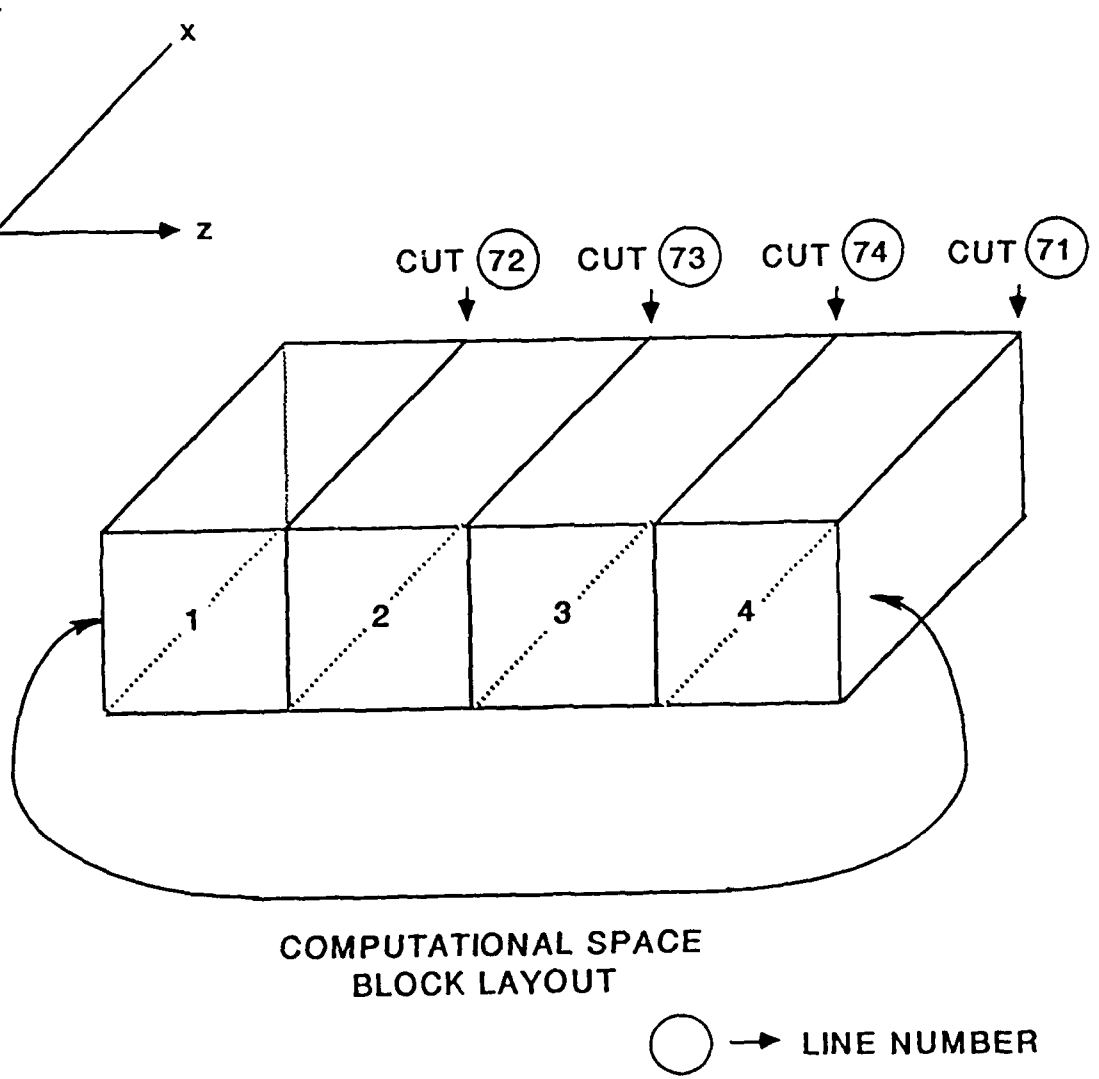


Figure 78. Four-Block Layout

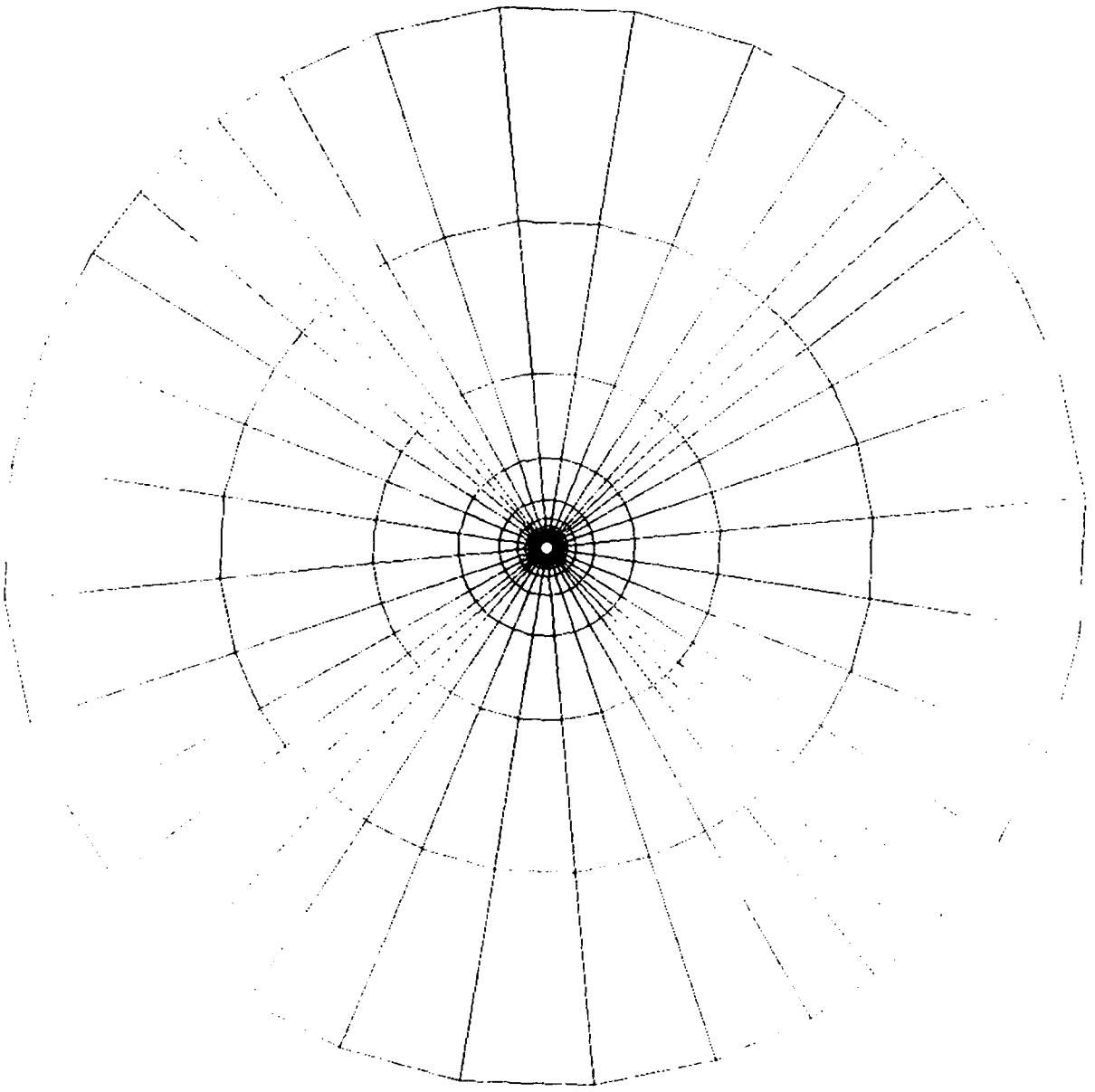


Figure 79. Front View (4-Block Grid)

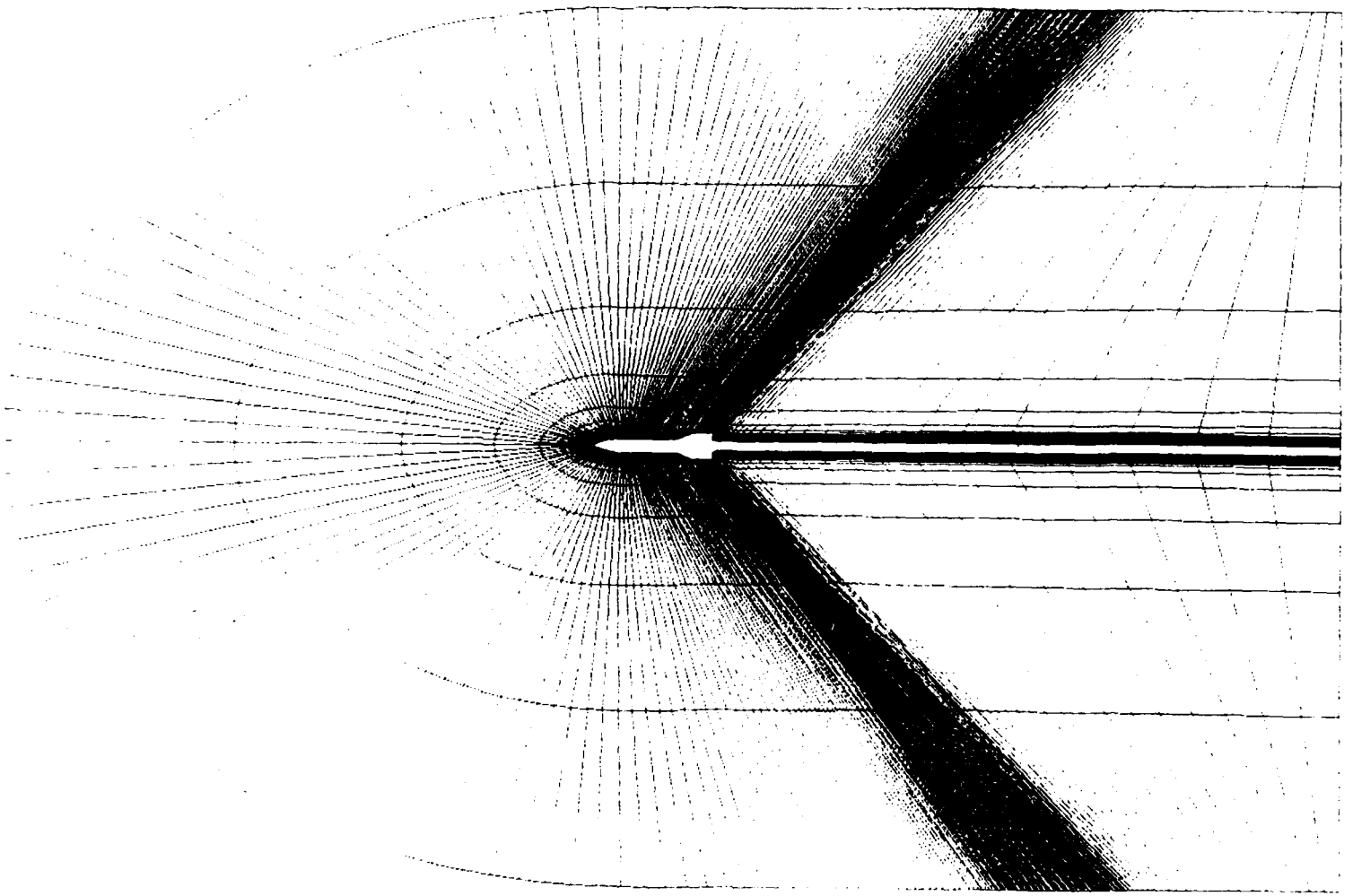


Figure 80. Side View (4 Block Grid)

```
SINPUT ITEM = "STORE",RSTORE="FILE", FILE=72,ITMAX=1,TOL=1.0E-06,ACCPAR=.0,
   OUTER="NO", CHECK="NO" $
SINPUT ITEM = "INITIAL",CONTYP="INITIAL",BLEND="ARC","ARC","LINEAR",
   ALL="YES"  $
SINPUT ITEM = "BLOCK", SIZE=140,24,10$
SINPUT ITEM = "FILE", START= 1, 1, 1, END=140, 1,10, CLASS="FIX",
   FILE=50,REWIND="YES"$
SINPUT ITEM = "FILE", START= 1,24, 1, END=140,24,10, CLASS="FIX",
   FILE=50,REWIND="NO"$
SINPUT ITEM = "FILE", START=140, 1, 1, END=140,24,10, CLASS="FIX",
   FILE=50,REWIND="NO"$
SINPUT ITEM = "FILE", START= 1, 1, 1, END= 1,24,10, CLASS="FIX",
   FILE=50,REWIND="NO"$
SINPUT ITEM = "FILE", START=68,16, 1, END=68,24, 1, FILE=50,
   REWIND="NO"$
SINPUT ITEM = "FILE", START=113,16, 1, END=113,24, 1, FILE=50,
  REWIND="NO"$
SINPUT ITEM = "FILE", START=68,16,10, END=68,24,10, FILE=50,
  REWIND="NO"$
SINPUT ITEM = "FILE", START=113,16,10, END=113,24,10, FILE=50,
  REWIND="NO"$
SINPUT ITEM = "FILE", START=68, 1, 1, END=113,16, 1, CLASS="FIX",
  FILE=50,REWIND="NO"$
SINPUT ITEM = "FILE", START=68, 1,10, END=113,16,10, CLASS="FIX",
  FILE=50,REWIND="NO"$
SINPUT ITEM = "INTERP", BLEND="ARC","ARC", START= 1, 1, 1, END=68,24, 1S
SINPUT ITEM = "INTERP", BLEND="ARC","ARC", START=68,16, 1, END=113,24, 1S
SINPUT ITEM = "INTERP", BLEND="ARC","ARC", START=113, 1, 1, END=140,24, 1S
SINPUT ITEM = "INTERP", BLEND="ARC","ARC", START= 1, 1,10, END=68,24,10S
SINPUT ITEM = "INTERP", BLEND="ARC","ARC", START=68,16,10, END=113,24,10S
SINPUT ITEM = "INTERP", BLEND="ARC","ARC", START=113, 1,10, END=140,24,10S
SINPUT ITEM = "FIX" , START=68,1,1 , END=113,16,1 $
SINPUT ITEM = "FIX" , START=68,1,10 , END=113,16,10 $
SINPUT ITEM = "BLOCK", SIZE=140,24,10$
SINPUT ITEM = "FILE", START= 1, 1, 1, END=140, 1,10, CLASS="FIX",
   FILE=60,REWIND="YES"$
SINPUT ITEM = "FILE", START= 1,24, 1, END=140,24,10, CLASS="FIX",
   FILE=60,REWIND="NO"$
SINPUT ITEM = "FILE", START=140, 1, 1, END=140,24,10, CLASS="FIX",
   FILE=60,REWIND="NO"$
SINPUT ITEM = "FILE", START= 1, 1, 1, END= 1,24,10, CLASS="FIX",
   FILE=60,REWIND="NO"$
SINPUT ITEM = "FILE", START=68,16, 1, END=68,24, 1, FILE=60,
   REWIND="NO"$
SINPUT ITEM = "FILE", START=113,16, 1, END=113,24, 1, FILE=60,
  REWIND="NO"$
SINPUT ITEM = "FILE", START=68,16,10, END=68,24,10, FILE=60,
  REWIND="NO"$
SINPUT ITEM = "FILE", START=113,16,10, END=113,24,10, FILE=60,
  REWIND="NO"$
SINPUT ITEM = "FILE", START=68, 1, 1, END=113,16, 1, CLASS="FIX",
  FILE=60,REWIND="NO"$
SINPUT ITEM = "FILE", START=68, 1,10, END=113,16,10, CLASS="FIX",
  FILE=60,REWIND="NO"$
SINPUT ITEM = "INTERP", BLEND="ARC","ARC", START= 1, 1, 1, END=68,24, 1S
SINPUT ITEM = "INTERP", BLEND="ARC","ARC", START=68,16, 1, END=113,24, 1S
SINPUT ITEM = "INTERP", BLEND="ARC","ARC", START=113, 1, 1, END=140,24, 1S
SINPUT ITEM = "INTERP", BLEND="ARC","ARC", START= 1, 1,10, END=68,24,10S
SINPUT ITEM = "INTERP", BLEND="ARC","ARC", START=68,16,10, END=113,24,10S
SINPUT ITEM = "INTERP", BLEND="ARC","ARC", START=113, 1,10, END=140,24,10S
SINPUT ITEM = "FIX" , START=68,1, 1 , END=113,16,1 $
SINPUT ITEM = "FIX" , START=68,1,10 , END=113,16,10 $
SINPUT ITEM = "BLOCK", SIZE=140,24,10$
SINPUT ITEM = "FILE", START= 1, 1, 1, END=140, 1,10, CLASS="FIX",
   FILE=70,REWIND="YES"$
SINPUT ITEM = "FILE", START= 1,24, 1, END=140,24,10, CLASS="FIX",
```

```
   FILE=70,REWIND="NO"$
$INPUT ITEM = "FILE", START=140, 1, 1, END=140,24,10, CLASS="FIX",
   FILE=70,REWIND="NO"$
$INPUT ITEM = "FILE", START= 1, 1, 1, END= 1,24,10, CLASS="FIX",
   FILE=70,REWIND="NO"$
$INPUT ITEM = "FILE", START=68,16, 1, END=68,24, 1, FILE=70,
   REWIND="NO"$
$INPUT ITEM = "FILE", START=113,16, 1, END=113,24, 1, FILE=70,
  REWIND="NO"$
$INPUT ITEM = "FILE", START=68,16,10, END=68,24,10, FILE=70,
  REWIND="NO"$
$INPUT ITEM = "FILE", START=113,16,10, END=113,24,10, FILE=70,
  REWIND="NO"$
$INPUT ITEM = "FILE", START=68, 1, 1, END=113,16, 1, CLASS="FIX",
  FILE=70,REWIND="NO"$
$INPUT ITEM = "FILE", START=68, 1,10, END=113,16,10, CLASS="FIX",
  FILE=70,REWIND="NO"$
$INPUT ITEM = "INTERP", BLEND="ARC","ARC", START= 1, 1, 1, END=68,24, 1$
$INPUT ITEM = "INTERP", BLEND="ARC","ARC", START=68,16, 1, END=113,24, 1$
$INPUT ITEM = "INTERP", BLEND="ARC","ARC", START=113, 1, 1, END=140,24, 1$
$INPUT ITEM = "INTERP", BLEND="ARC","ARC", START= 1, 1,10, END=68,24,10$
$INPUT ITEM = "INTERP", BLEND="ARC","ARC", START=68,16,10, END=113,24,10$
$INPUT ITEM = "INTERP", BLEND="ARC","ARC", START=113, 1,10, END=140,24,10$
$INPUT ITEM = "FIX" , START=68,1, 1 , END=113,16, 1 $
$INPUT ITEM = "FIX" , START=68,1,10 , END=113,16,10 $
$INPUT ITEM = "BLOCK", SIZE=140,24,10$
$INPUT ITEM = "FILE", START= 1, 1, 1, END=140, 1,10, CLASS="FIX",
   FILE=80,REWIND="YES"$
$INPUT ITEM = "FILE", START= 1,24, 1, END=140,24,10, CLASS="FIX",
   FILE=80,REWIND="NO"$
$INPUT ITEM = "FILE", START=140, 1, 1, END=140,24,10, CLASS="FIX",
   FILE=80,REWIND="NO"$
$INPUT ITEM = "FILE", START= 1, 1, 1, END= 1,24,10, CLASS="FIX",
   FILE=80,REWIND="NO"$
$INPUT ITEM = "FILE", START=68,16, 1, END=68,24, 1, FILE=80,
   REWIND="NO"$
$INPUT ITEM = "FILE", START=113,16, 1, END=113,24, 1, FILE=80,
  REWIND="NO"$
$INPUT ITEM = "FILE", START=68,16,10, END=68,24,10, FILE=80,
  REWIND="NO"$
$INPUT ITEM = "FILE", START=113,16,10, END=113,24,10, FILE=80,
  REWIND="NO"$
$INPUT ITEM = "FILE", START=68, 1, 1, END=113,16, 1, CLASS="FIX",
  FILE=80,REWIND="NO"$
$INPUT ITEM = "FILE", START=68, 1,10, END=113,16,10, CLASS="FIX",
  FILE=80,REWIND="NO"$
$INPUT ITEM = "INTERP", BLEND="ARC","ARC", START= 1, 1, 1, END=68,24, 1$
$INPUT ITEM = "INTERP", BLEND="ARC","ARC", START=68,16, 1, END=113,24, 1$
$INPUT ITEM = "INTERP", BLEND="ARC","ARC", START=113, 1, 1, END=140,24, 1$
$INPUT ITEM = "INTERP", BLEND="ARC","ARC", START= 1, 1,10, END=68,24,10$
$INPUT ITEM = "INTERP", BLEND="ARC","ARC", START=68,16,10, END=113,24,10$
$INPUT ITEM = "INTERP", BLEND="ARC","ARC", START=113, 1,10, END=140,24,10$
$INPUT ITEM = "FIX" , START=68,1, 1 , END=113,16, 1 $
$INPUT ITEM = "FIX" , START=68,1,10 , END=113,16,10 $
$INPUT ITEM="CUT",BLOCK=1,START=1,1,1,END=68,24,1,
                  IBLOCK=4,ISTART=1,1,10,IEND=68,24,10$
$INPUT ITEM="CUT",BLOCK=1,START=68,16,1,END=113,24,1,
                  IBLOCK=4,ISTART=68,16,10,IEND=113,24,10$
$INPUT ITEM="CUT",BLOCK=1,START=113,1,1,END=140,24,1,
                  IBLOCK=4,ISTART=113,1,10,IEND=140,24,10$
$INPUT ITEM="CUT",BLOCK=1,START=1,1,10,END=68,24,10,
                  IBLOCK=2,ISTART=1,1,1,IEND=68,24,1$
$INPUT ITEM="CUT",BLOCK=1,START=68,16,10,END=113,24,10,
                  IBLOCK=2,ISTART=68,16,1,IEND=113,24,1$
$INPUT ITEM="CUT",BLOCK=1,START=113,1,10,END=140,24,10,
                  IBLOCK=2,ISTART=113,1,1,IEND=140,24,1$
```

```
$INPUT ITEM="CUT",BLOCK=2,START=1,1,10,END=68,24,10,
                  IBLOCK=3,ISTART=1,1,1,IEND=68,24,1$
$INPUT ITEM="CUT",BLOCK=2,START=68,16,10,END=113,24,10,
                  IBLOCK=3,ISTART=68,16,1,IEND=113,24,1$
$INPUT ITEM="CUT",BLOCK=2,START=113,1,10,END=140,24,10,
                  IBLOCK=3,ISTART=113,1,1,IEND=140,24,1$
$INPUT ITEM="CUT",BLOCK=3,START=1,1,10,END=68,24,10,
                  IBLOCK=4,ISTART=1,1,1,IEND=68,24,1$
$INPUT ITEM="CUT",BLOCK=3,START=68,16,10,END=113,24,10,
                  IBLOCK=4,ISTART=68,16,1,IEND=113,24,1$
$INPUT ITEM="CUT",BLOCK=3,START=113,1,10,END=140,24,10,
                  IBLOCK=4,ISTART=113,1,1,IEND=140,24,1$
$INPUT ITEM = "END" $
$OUTPUT ITEM = "ERROR", BLKERR="YES" $
$OUTPUT ITEM = "END" $
```

MULTIPLE BODY CONFIGURATION

The EAGLE grid generator has the capability to model complex, multibody configurations. The configuration of the finned ogive-cylinder-ogive airframe has been presented earlier. Figure 81 illustrates the basic geometry of the body and the orientation of the three-body combination. The grid is constructed for one-half of the configuration to take advantage of symmetry and reduce subsequent computational costs of the flow solver. The general dimensions of the overall grid are 145 x 39 x 37 with over 275,000 total points. The configuration is divided with a vertical plane passing between the two upper bodies and through the lower body. The resulting plane is essentially treated as a reflection plane. Figure 1c shows a wireframe grid of the block arrangement with the reflection plane being the flat lower surface. The grid is made up of 30 blocks, arranged in three sections of 10 blocks each, stacked axially as in the wireframe grid. A frontal view of the 10-block arrangement in the fin area is shown in Figure 82a, b. Six of the blocks are built around the bodies; one block between each set of two fins. The remaining four blocks are field grid blocks extending to the outer boundary. A representative cross section of the elliptic grid generated in 50 iterations is presented in Figure 83. Figure 84 illustrates a side view of the grid through the upper whole body. The grid around the body is a C-grid that is immersed in the global H-grid. The algebraic grid is shown in 4a and the elliptic grid in 4b. The elliptic solver smooths the grid and interface between the C and H grids while keeping a tight, consistent grid immediately adjacent to the body. Generation of the 50 iteration elliptic grid took 687 CPU seconds on the Cray X-MP. Storage requirements were 2.9 million words of in-core memory and 8.22 million words of SSD memory.

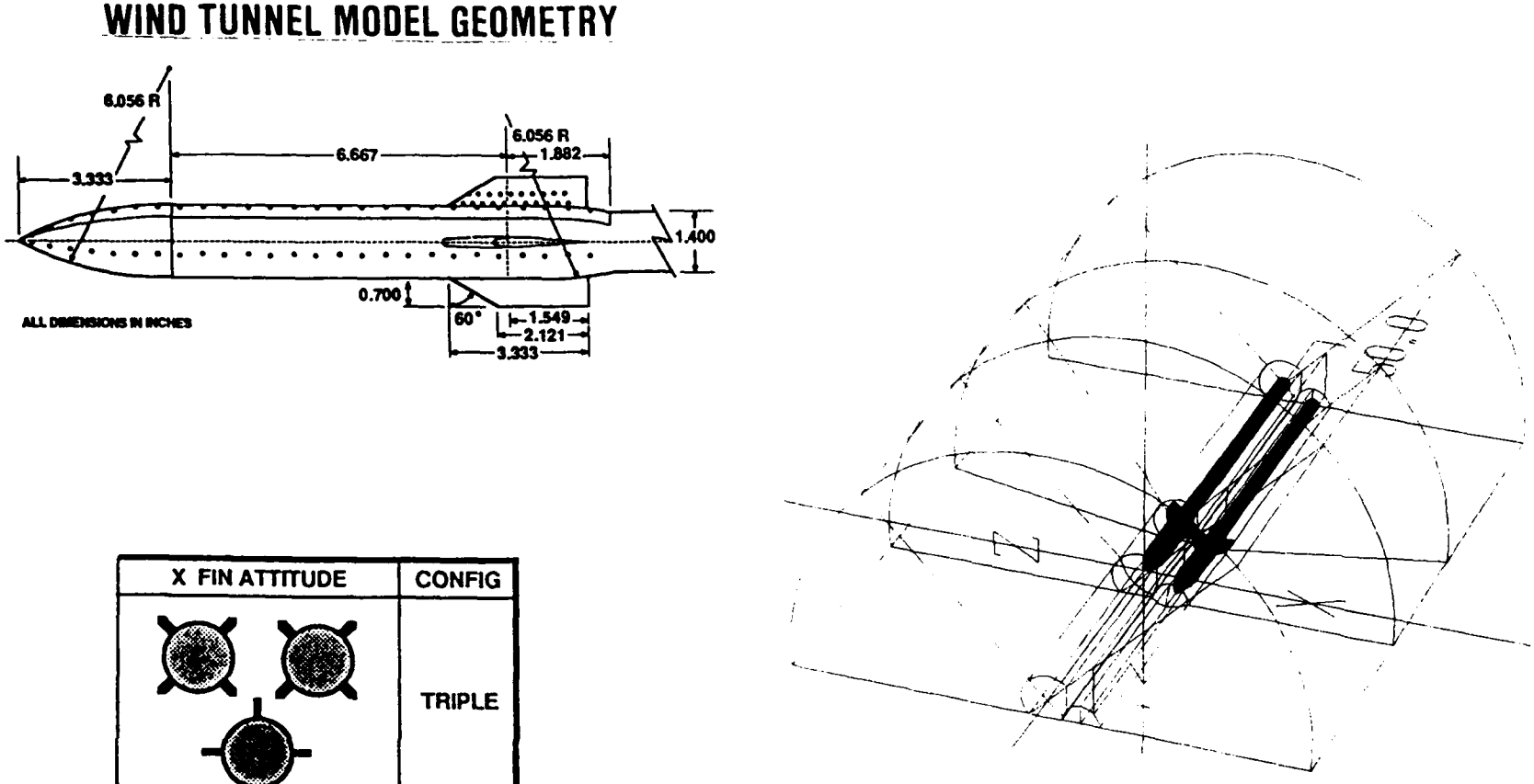
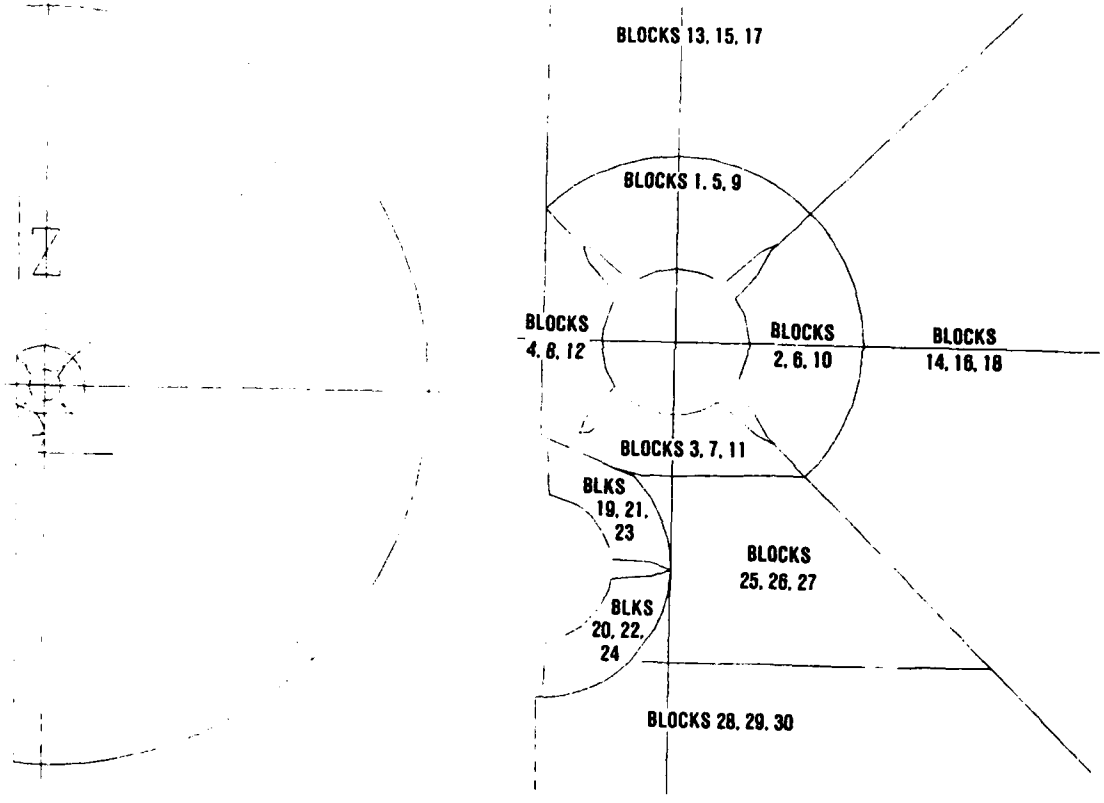


Figure 81. Three Finned Body Configuration



(a) (b)

Figure 82. 30-Block Grid Scheme

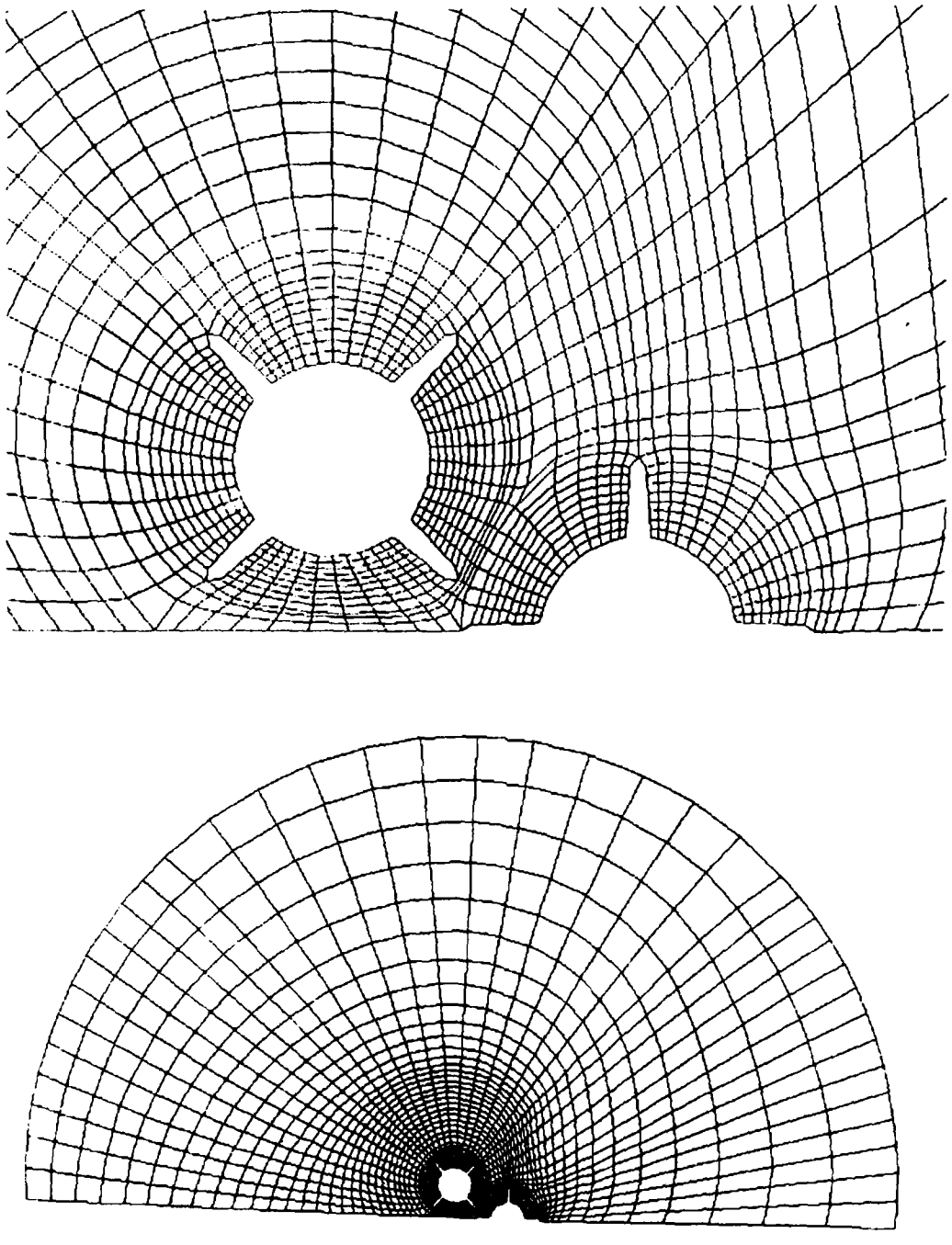


Figure 83. Elliptic Grid Cross Section, Frontal View

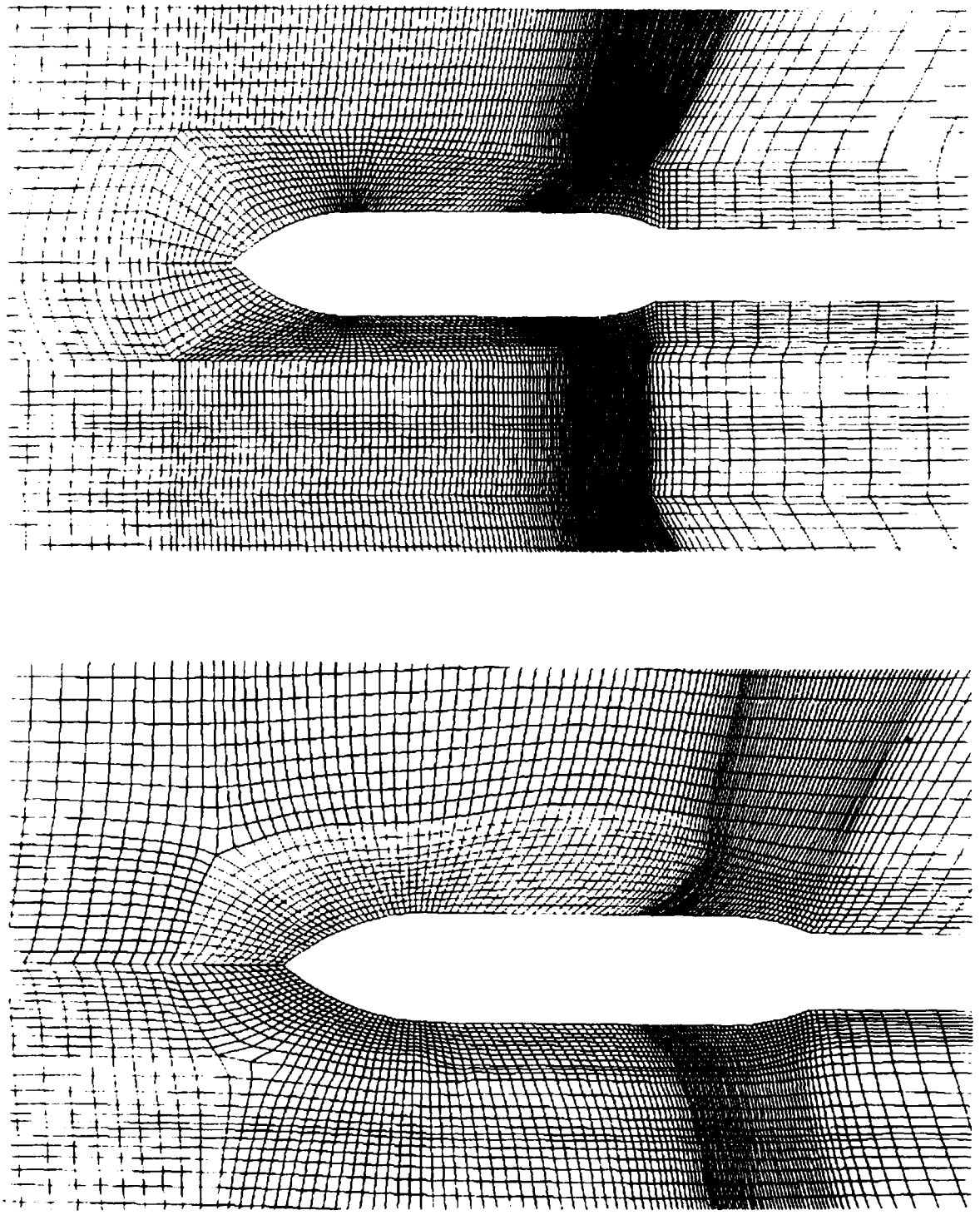


Figure 84. Algebraic Elliptic Grid Cross Section, Side View

Wing-Pylon-Store Grid

The grid developed for obtaining Euler solutions on the wing-pylon-store configuration is a 30-block system containing approximately 200,000 points. The overall structure, as depicted in Figure 85, is a C-O-type grid enclosing the pylon and store with an H-type grid system surrounding the wing and the C-O grid.

The grid defining the surfaces of the wing, pylon, and store was built using operations in the EAGLE boundary code (Figure 86). Coordinates for the wing root and tip were read in, and then the wing was built by interpolation. The pylon, store, and sting were constructed according to dimensions and specifications by building up curve segments and then rotating these curves, or interpolating between them. The pylon was affixed smoothly to the wing grid by inserting it into the lower wing surface through the INTSEC operation.

The boundary for the C-O grid, which encloses the entire pylon, store, and sting, was generated by rotating curves about an axis parallel to the store. Each of these curves included an arc segment emanating from a point ahead of the store nose and terminating at the leading edge of the wing at a point to the right or left of the pylon. A second segment of the rotated curves consisted of a spar line extracted from the lower wing grid, while the third part extended from the wing trailing edge to the trailing far-field grid boundary.

The C-O grid itself consists of five blocks. Embedded within this structure is a sixth block which fills the gap between the pylon and the upper surface of the store, as shown in Figure 87. A cylindrical grid structure, consisting of wedge-shaped quadrants, lies ahead of the C-O grid, extending from the leading edge of the wing to the upstream far-field boundary.

The remainder of the grid system, consisting of 20 blocks, surrounds the C-O grid and the cylindrical system. These remaining blocks are nearly rectangular except where they interface with the wing or curved segments of the store and pylon grid structure. Each of these 20 blocks was built up with operations in the EAGLE boundary code. The far-field boundaries were placed 40 store diameters upstream and downstream of the nose of the store, and 20 diameters outboard of the store.

The boundaries of the 30 blocks were used as input to the EAGLE elliptic grid generator. Several preliminary runs were made before the final block

boundary configuration was arrived at. These preliminary runs were necessary due to the unusual angles of intersection between the C-O grid and the lower surface of the wing. The final code run was for 50 iterations of the elliptic grid solver, requiring 526 CPU seconds on a Cray XMP/24 and nearly 24 million SSD blocks moved.

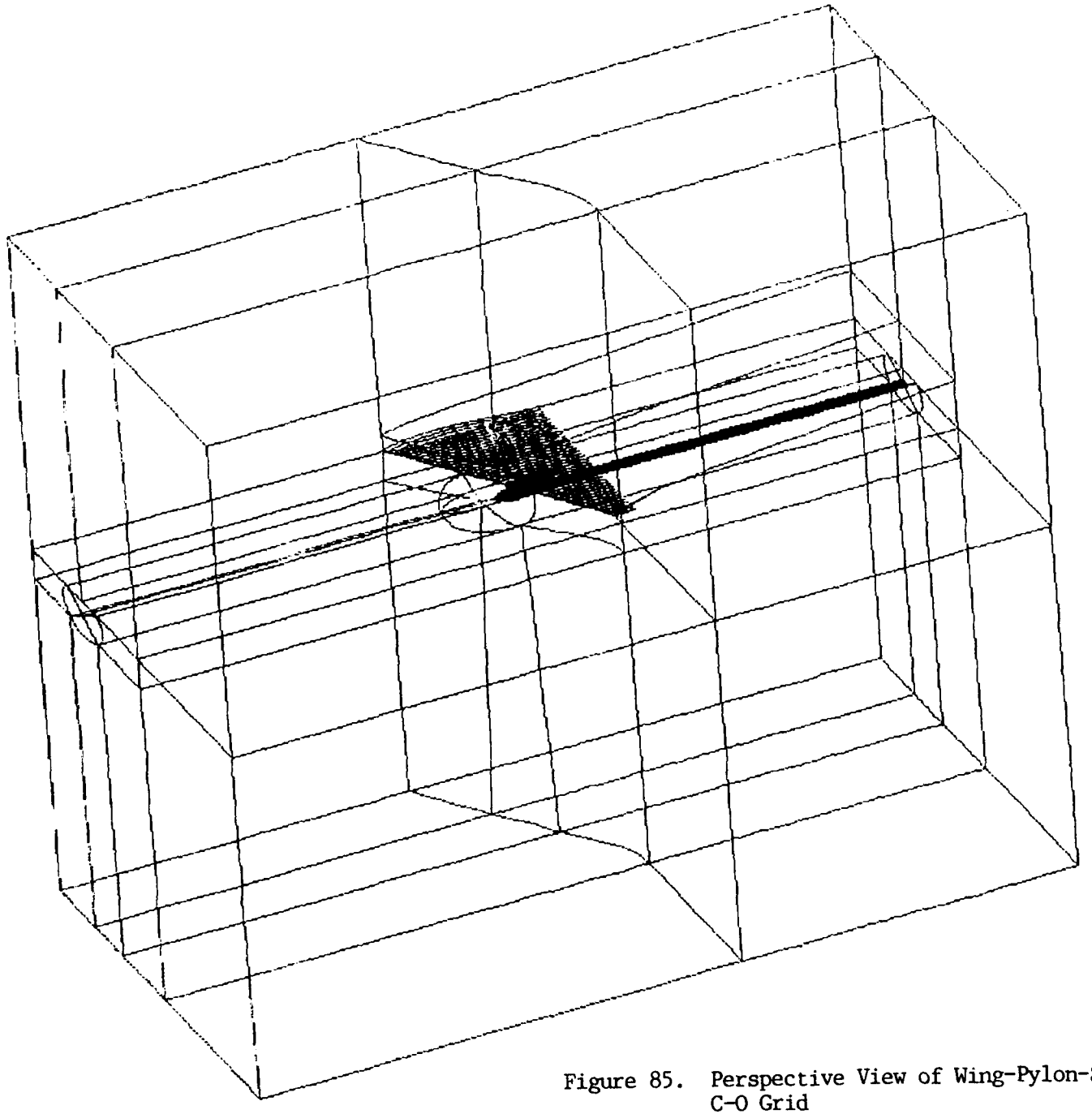


Figure 85. Perspective View of Wing-Pylon-Store, C-O Grid

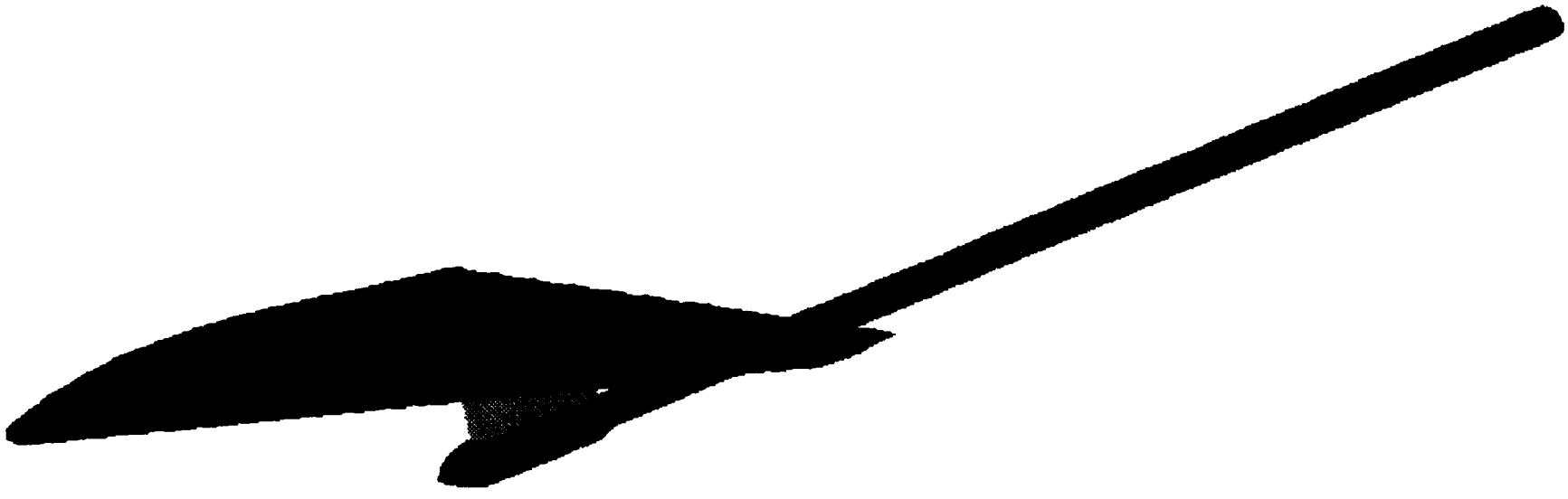


Figure 86. Wing with Pylon, Store, and Sting

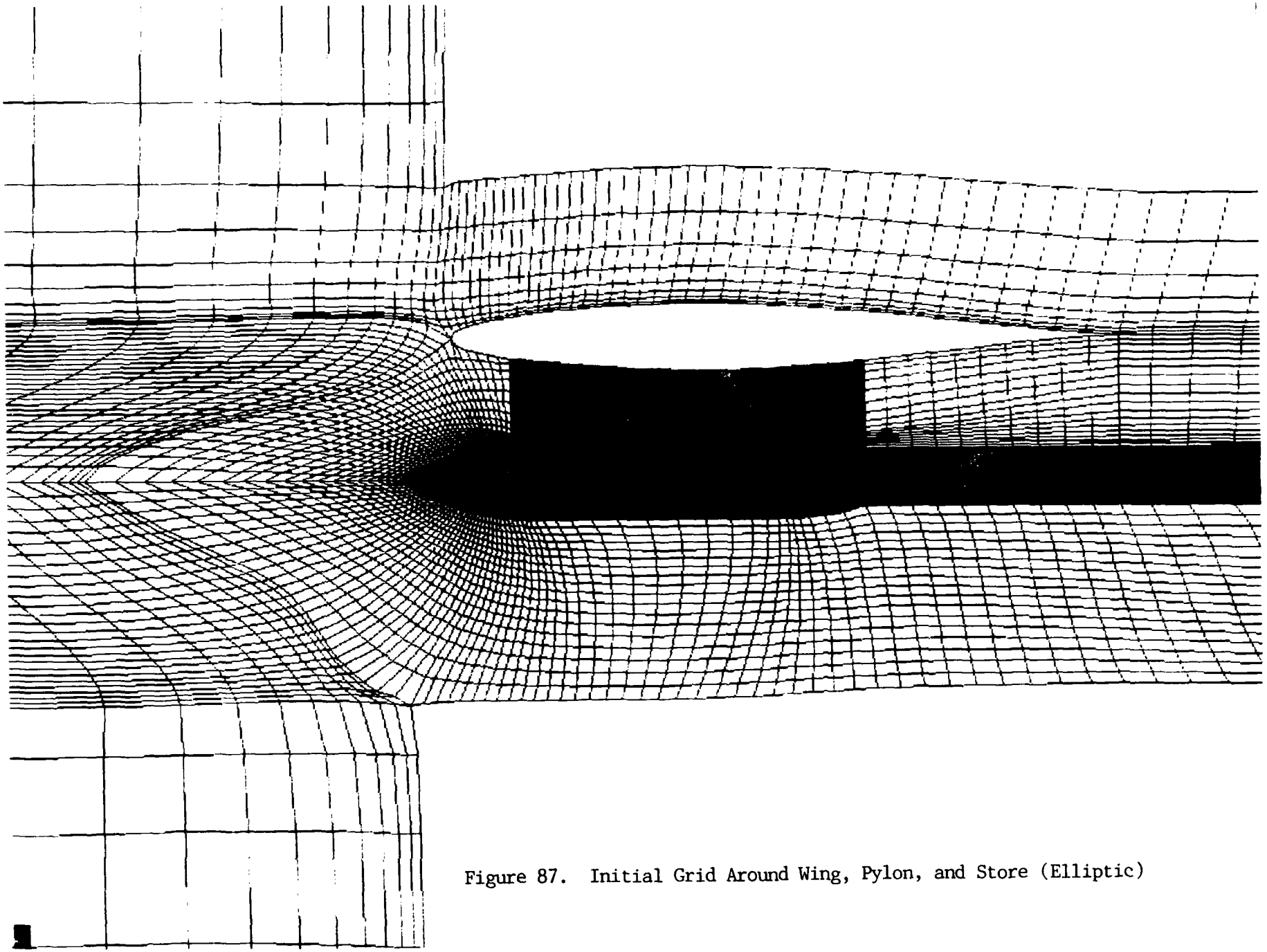


Figure 87. Initial Grid Around Wing, Pylon, and Store (Elliptic)

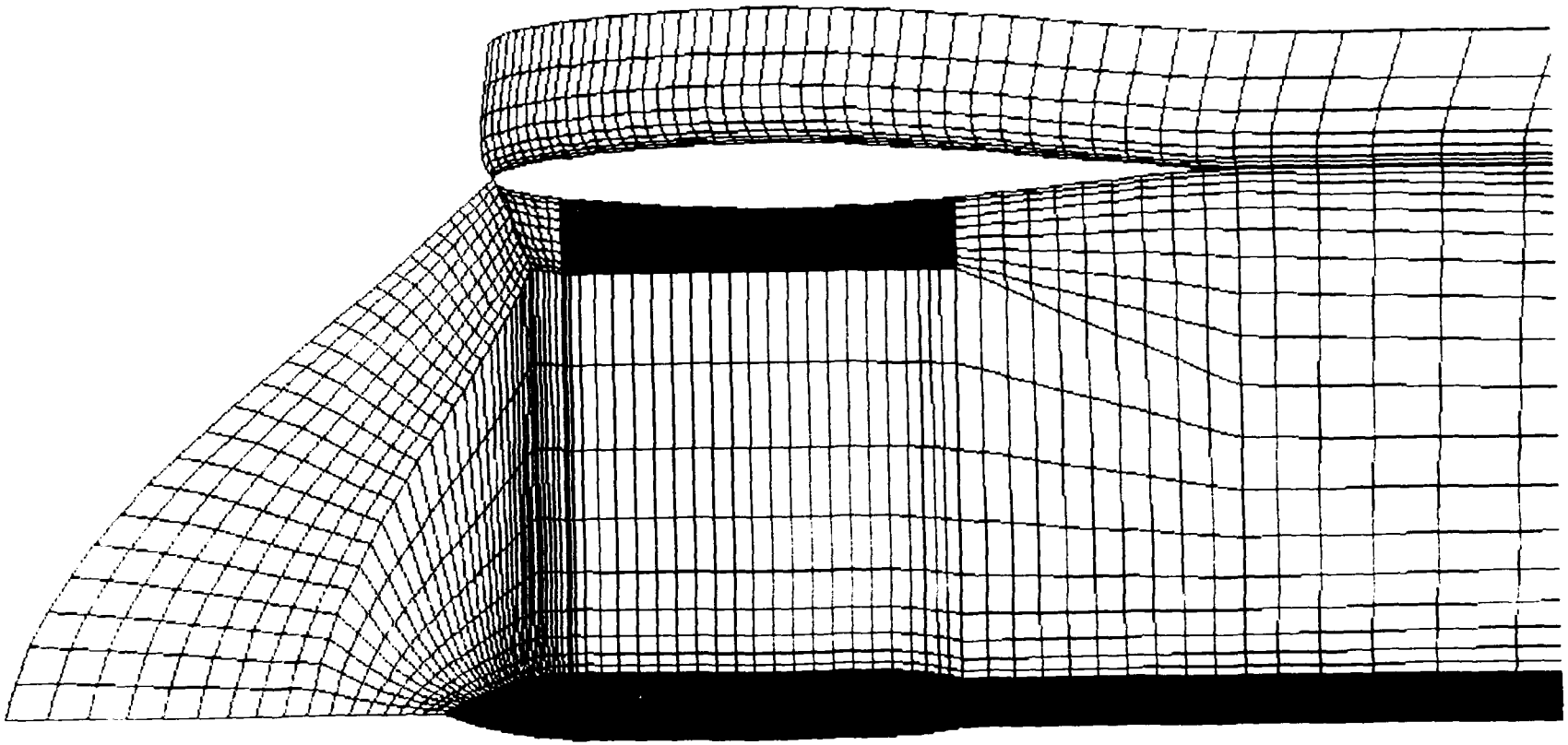


Figure 88. Separated Grid Around Wing, Pylon, and Store (Algebraic)